CMIF
Center for Multisource Information Fusion

Research on an Enabling Infrastructure for Distributed Simulation

Kai Harth

Report No. CMIF 3-01

March 2001

Acknowledgement: This research project was supported by Grant F49620-99-1-0090 from the Air Force Office of Scientific Research (AFOSR) and technical direction was provided by staff from the Air Force Flight Test Center (AFFTC) at Edwards AFB, California; the Center for Multisource Information Fusion is grateful to both agencies and their staffs for the support and guidance associated with this project.

CUBRC

CALSPAN–UNIVERSITY AT BUFFALO RESEARCH CENTER, INC.

University at Buffalo
The State University of New York

REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-

0386

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2001	Final Report (1/1/99 – 12/31/00)

4. TITLE AND SUBTITLE
Research on an Enabling Infrastructure for Distributed Simulation

6. FUNDING NUMBERS
G- F49620-99-1-0090

5. AUTHORS
Kai Harth

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Calspan-UB Research Center, Inc.
P.O. Box 400
4455 Genesee Street
Buffalo, New York 14225

8. PERFORMING ORGANIZATION REPORT NUMBER
CMIF 3-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
801 N. Randolph Street, Room 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Pubilcly releasable; unlimited distribution

13. ABSTRACT *(Maximum 200 words)*
Distributed simulation plays an important role for the military modeling and simulation community. Various reasons support the integration of independent simulators into distributed simulation federations. Among these are the increased computational power becoming available, and the possibility of reusing already existing sophisticated software simulations for new purposes, ie a leveraging motivation.The US Department of Defense has devised its own standard for simulation reuse and interoperability, the High Level Architecture (HLA).The purpose of this report is firstly to analyze some of the issues that are related to distributed simulation in general. These issues include, for example, concepts of simulation timing or issues of simulation interoperability. Secondly, a close look has to be taken at the HLA and its implementation, the Run Time Infrastructure RTI, to get a better understanding of the capabilities and the functioning of this tool. One goal of the effort reported on is to evaluate the HLA's particular applicability for the simulation of distributed data fusion and for military development, testing and evaluation (DT&E). Thirdly, a concept is devised for making use of the rather-complex HLA/RTI tool for small-scale university-type distributed simulation research. This is done by implementing an easy-to-use programming environment, based on HLA, and necessary control and support tools. In this report, this is called the "CMIF HLA Environment", as the work was conducted at the State University of New York's "Center for Multisource Infromation Fusion or CMIF.As the last part of this work, a first proof of concept experiment/demo is done to validate the implemented software. This is realized by constructing a demo application, a distributed simulation experiment with several components that make use of all the features and mechanisms of the CMIF HLA Environment.

14. SUBJECT TERMS
Distributed Simulation, Distributed Data Fusion, Laboratory Software

15. NUMBER OF PAGES
214

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
U	U	U	UL

RESEARCH ON AN ENABLING INFRASTRUCTURE FOR DISTRIBUTED SIMULATION

by

Kai Harth

February 1, 2001

A thesis submitted to the
Faculty of the Graduate School of the
State University of New York at Buffalo
in partial fulfillment of the requirements for the
degree of
Master of Science

Department of Mechanical and Aerospace Engineering

Acknowledgements

My pursuing of the degree of Master of Science would have been out of question without the help and support of many generous people. I am deeply grateful. The following list is without order and makes no claim to completeness:

The German Fulbright Commission, for giving me the chance to come to the United States for graduate studies in the first place.

Professor James Llinas, my advisor. Despite his workload, he never failed to support me during eventual technical and motivational difficulties. His funding gave me the chance to extend my stay and even return to Buffalo twice. The CMIF lab became something like a second home.

The Department of Mechanical and Aerospace Engineering and its faculty, for providing an excellent and challenging academic environment.

My committee members, Professor Crassidis and Professor Mook, for taking their time to read through all of this.

My family and friends, for always encouraging and supporting me, and for letting me go at all.

Contents

List of Figures

Abstract

Chapter 1

Introduction

1.1 Background

The importance of computers, especially computer simulation, in today's technical world is continously increasing. It is widely accepted, that the use of computer simulation can save valuable time and costly resources. This is also true for the realm of military applications, where simulations are used over a wide range, from training up to research and development (R&D). Over time, two situations have evolved: Firstly, simulation scenarios tend to become more and more complex. And secondly, a large number of sophisticated and specialized simulators are already available and a reuse of these software progams is desirable. This lead to the field of distributed simulation, where different simulation programs are interconnected and cooperate with each other to form a new simulation application. Although the approach of distributed simulation is promising and offers a solution to both problems –increasing

complexity and the desire for reuse– it also raises a lot of new questions and issues: Interface and communication standards are needed and general aspects of simulation interoperability need to be dealt with.

This work is part of the research efforts of the Center for Multisource Information Fusion, CMIF. The CMIF, which is headed by Professor James Llinas, Ph.D., is a research facility at the State University of New York at Buffalo. Part of the research, which is done, is concerned with distributed data fusion, a field that is closely related to that of distributed simulation.

The data fusion research has lead to a close cooperation between the CMIF and the US military, more specifically, the Air Force Electronic Warfare Flight Test Range Facility at Edwards Air Force Base. There, Test and Evaluation (T&E) experiments are conducted with electronic warfare components, on in-flight platforms as well as in laboratory test-stands. The test ranges –Electronic Combat Range and Nellis Range Complex– are instrumented and populated with high-fidelity, manned or unmanned threat simulators. Additional emitter-only threat simulators are used to provide high signal density, typical of operational electronic warfare environments.

The US military has its own standards for distributed simulation. Two older ones, DIS[1] and ALSP[2] are being phased out now. Their successor is the High Level Architecture (HLA) which features new and improved capabilities. In this treatise, the HLA will be analyzed and then a concept will be devised how to utilize the HLA in the form of a lab environment for distributed data fusion research at CMIF. Also, the implementation of this concept, the "CMIF HLA Environment", will be introduced.

[1]DIS–Distributed Interactive Simulation

[2]ALSP–Aggregate Level Simulation Protocol

1.2 Purpose of this work

The initial motivation for this project was to analyze future scenarios for the Edwards AFB Range Testing Facility. The question is: How will distributed simulation contribute to and enhance testing of electronic warfare components? To be able to answer this, it was first necessary to become more familiar with the general aspects of distributed simulation. The High Level Architecture (HLA) then came into focus because it is the mandatory framework for all future military simulations. After a closer look was taken at the HLA and its implementation, the Run Time Infrastructure, it became clear that this was a tool with great capabilities. It would also be desirable to utilize it for the distributed data fusion research done by CMIF in general. Thus the demand for a programming environment for the CMIF lab arose and the main project focus shifted to the concept-development, implementation and testing of the "CMIF HLA Environment".

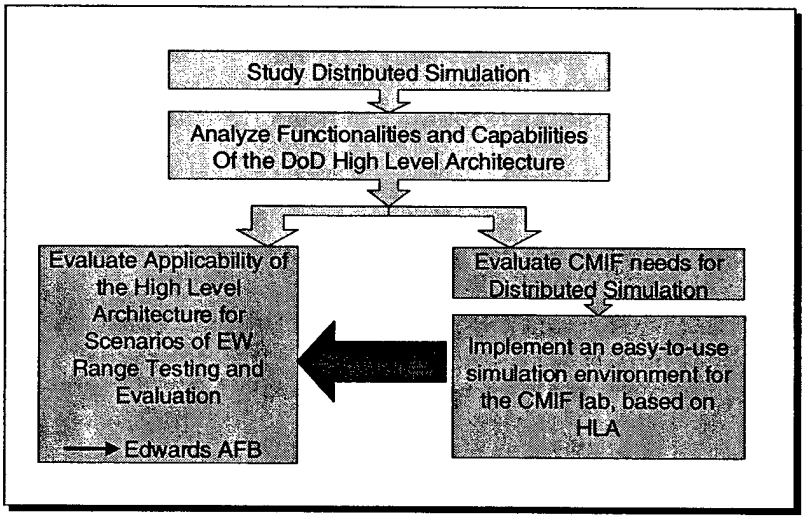

Figure 1.1: Purpose of this work: Two separate branches evolve but mutual benefits exist.

As depicted in figure 1.1, this leads to separate outcomes of this project which are not too apparently related to each other. The to be implemented lab environment for distributed simulation will be tailored to the needs of CMIF. Not neccessarily will it be applicable in other settings too. Nevertheless, the hands-on experience gained in dealing with distributed simulation in general and the Department of Defense High Level Architecture in particular will definitely be valuabe for further studies. Thus, the relevance of this research, and even the CMIF software, for the work done at Edwards AFB becomes evident.

The proceeding of this project was not fixed from the very beginnig, but rather did the work evolve over time, after more and more of the aspects and problems that are related to the High Level Architecture and to the distributed simulation field became clear.

Chapter 2

Issues of Distributed Simulation

In the following, after discussing the motivation for distributed simulation in general, several issues related to the overall field will be introduced. At this time, they might seem more or less unrelated to each other. But for the subsequent chapters, these topics will be needed as background information.

2.1 Motivation for Distributed Simulation

Previous research done by CMIF has already dealt with issues of distributed simulation. Software was implemented to simulate and study the behaviour of distributed data fusion nodes, multi-agent teams and the like. But always, the aspect of distribution itself had only been emulated. This meaning, the software "pretended" to deal with separate and independent components while, in fact, the whole simulation was carried out in one single computer program, one single process.

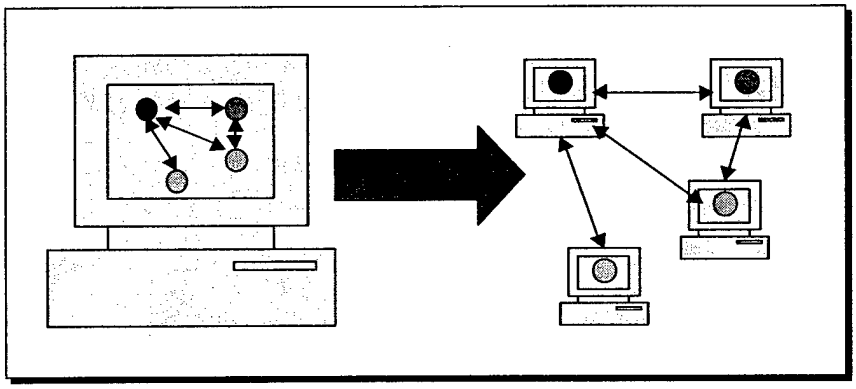

Figure 2.1: The step from emulated distributed simulation to "real" distributed simulation

Now the next logical step would be to try to do similar experiments in a "real" distributed environment, as it is suggested in figure 2.1. The components of one simulation experiment would run in various processes on different computers, connected by a local area network (LAN). But obviously this will introduce a number of disadvantages. Mainly, making this transition will lead to increased complexity due to

- the neccessity of dealing with inter-process and inter-platform communications and
- synchronization and coordination issues between components.

At the same time though, also advantages arise with the transition to simulation that is in fact physically distributed:

- Increased computational power becomes available by utilizing the resources of several computers at a time.

- Due to the modular character, the reuse of components (especially support tools etc.) becomes possible and will, after an initially increased effort, make development more effective from project to project.

- Dealing with the additional problems of distributed simulation, and the resulting experience gain, will lead to knowledge advantages for subsequent work.

This makes clear, that the mentioned disadvantages can at the same time be viewed as positive aspects. And anyway, since real world systems are always distributed in one respect or another, there is no use in trying to get around the difficult issues that distribution entails. On the contrary: It is desirable to gain the expertise in dealing with distributed simulation while working in a small, manageable environment. Those lessons learned will then be readily available for subsequent and more advanced projects.

2.2 The Role of Time

When simulation components are implemented, which are supposed to work together in some kind of distributed simulation, among other things, a timing scheme has to be agreed upon. Timing can be handled in several different ways (ascending order of complexity):

Time is irrelevant. This would be true rather for technical simulations like FEM.

Time is relevant, but synchronization is only internal That means, although time is an integral part of the simulation behaviour, simulation time bears no

relation to real-time. Also, the simulation-speed or -rate can vary at the simulation's own discretion. An example would be the Canadian military simulation software CASE_ATTI [Cas00], which is also used in the CMIF lab. It features time synchronization between its internal components, but proceedes with the simulation execution always as fast (or slow) as system performance allows it.

Synchronized to time scale but slower than real-time: Each time step in the simulation is of equal length if measured in real-time, but simulation time proceeds slower.

Real-time synchronized: The simulation is fully following the real-time scale. This concept is of special importance, because only in this case can "Live Players" or real components be integrated into the otherwise virtual simulation environment[1].

Faster than real-time: If computing resources allow it, "fast-forwarding" through a simulation saves time, of course.

Figure 2.2 depicts these different approaches. It also introduces a scaling factor that relates the last three possibilities to each other, using the following formula:

$$Sc = \frac{t_{sim}}{t_{real}}$$

where Sc represents a scaling factor. If $Sc < 1$, we get slower than real-time behaviour, $Sc = 1$ represents real-time synchronization, and $Sc > 1$ leads to "fast forward".

[1]Exceptions exist: Sometimes pilot training simulators are operated faster than real-time to induce a higher level of stress to the tested person.

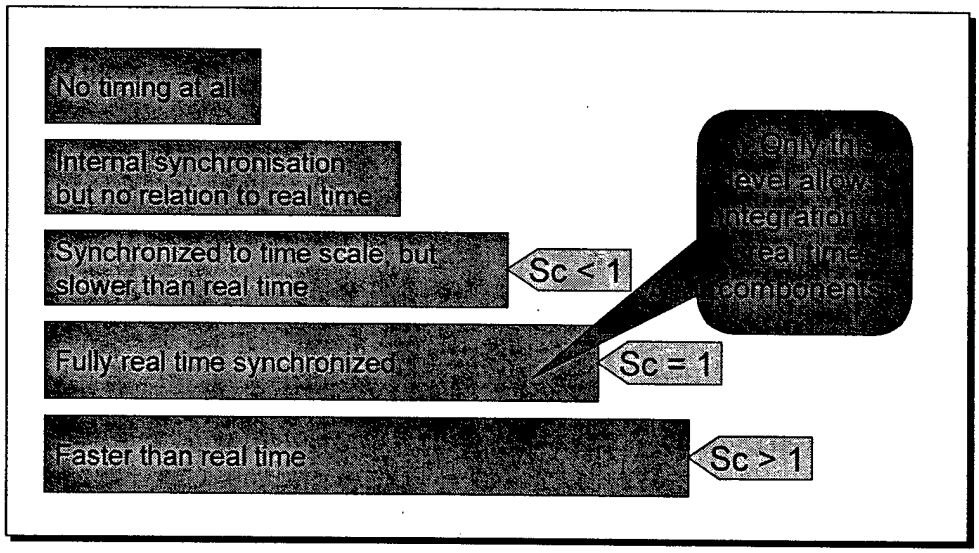

Figure 2.2: Overview over different timing concepts

Before finally applying one of these three concepts for a distributed simulation, it has to be made sure, that the chosen scaling factor Sc is feasible for the given simulation workload and system resources. Otherwise, a simulation component might fall behind, timing might be lost and simulation results would then be rendered useless.

Another technical aspect related to simulation timing, is the question of timing control. It has to be determined, if one component will be responsible for the timing of others ("Master-Slave") or if each component takes care of timing itself.

2.3 Multi-Threading

Another important aspect that arises when dealing with advanced simulation, is *multi-threading*. In the programming world, the term thread is used for a part of a program that progresses with its flow of control basically on its own. Obviously, the term

became necessary only after computer systems and advanced programming languages became able to deal with more than one thread at a time. This is where multi-threading comes into play. Figures 2.3 and 2.4 show the basic difference: A computer program has to complete different tasks, and in the single-thread situation, this has to be done one task after the other. In the multi-threaded setting, in contrast, each task can be worked on in its own thread of execution.

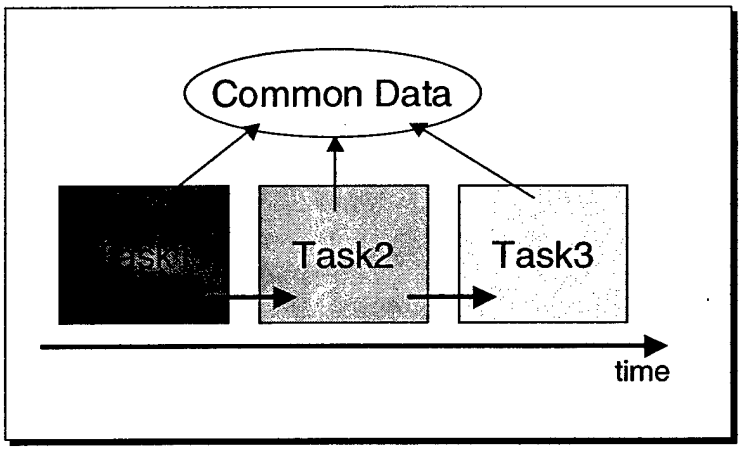

Figure 2.3: Depiction of a single thread program execution

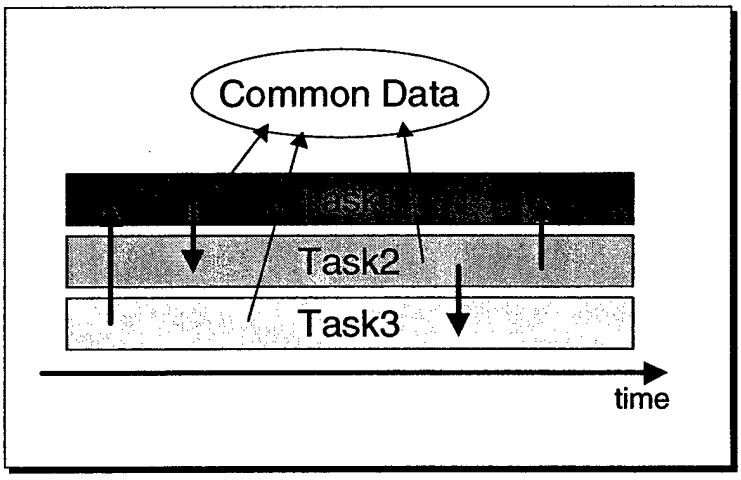

Figure 2.4: A program with multiple threads executing in parallel.

Threads can be used on a user-level, i. e. in application development, as well as on a system level (kernel threads).

The main advantages that multi-threading provides over single threads, are the exploitation of latency and concurrency. This means, that while with a single thread inherent waiting time of different tasks will add up, with multi-threading, latencies in one thread will be utilized for the execution of another task.

It must be mentioned, that multiple threads on a "normal" computer are never really parallel. They are just broken down into small segments on the system level. The execution of these segements is then switched between those belonging to different threads, making the tasks appear to be worked on in parallel. Of course, this is different with parallel-processor computers and with parallel computing systems.

As the two graphs suggest, difficulties with multi-threading arise because in most cases, threads are never totally independent from each other. They might access a pool of common data or directly interact with each other. The following problems come into play:

Data consistency: When different threads manipulate the same data, extreme caution caution has to be exercised. It has to be defined who is allowed to do what and at which point in time.

Dead-locks: Usually threads might interact with each other and thus affect their respective behaviour. Careful considerations have to be made about all possible states that a program's task might be in when an interaction is attempted, to avoid illogical behaviour or dead ends in the execution. For example: Thread

one is waiting for input from thread two to proceed, and vice versa.

The mention of all these issues is not only important because the Run Time Infrastructure itself is highly multi-threaded and also the implementation of the CMIF HLA Environment will make use of several threads. Similarily, the concepts, advantages and disadvantages also apply to a multi-process situation and even to a multi-computer scenario of distributed simulation. In these cases, the same considerations about data consistency and possible dead-locks have to be made to ensure a smoothly performing system. That means, although the application developer who chooses to use the RTI and the CMIF HLA Environment maybe does not have to worry too much about internal threading issues of the provided software, he still has to keep in mind the same lurking problems and plan ahead to avoid them, also on an application level.

2.4 Time-Stepped versus Event-Based

This section addresses another central aspect of computer simulation. The question in this case is: What is the driving factor behind a simulation's proceeding, what stimulates the simulation to do its actual work? This leads to the following distinction[2]:

"Time-Stepped": In this case, time is the driving factor behind the simulation. Technically, the time is progessing in certain increments, and within each step, certain tasks are processed.

[2]The two terms are taken from the HLA terminology.

"Event-Based": External input in the form of events is driving the simulation. That means, the simulation reacts rather than acts.

In reality, simulation applications represent in most cases a hybrid combination of both behaviours. For example, a military flight simulator which mainly progesses with the simulation along the time-line, but also reacts to event-like pilot input or external events like "collision" or "missile fired".

This way of looking at simulation behaviour is not necessarily limited to distributed simulation. But it is with distributed simulation, especially in the case of hybrid behaviour, that matters can get very complicated.

Chapter 3

The Department of Defense High Level Architecture

The best way to start an introduction to the High Level Architecture is to quote directly from the Department of Defense (DoD):

> "The High Level Architecture (HLA) is a general purpose architecture for simulation reuse and interoperability. It was developed under the leadership of the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the large number of different types of simulations developed and maintained by the DoD." [Def00b]

In what follows, a synopsis to the HLA will be given. Because plenty of detailed documentation about the HLA is available, and because a simple repetition of all this material was not desired, the overview will be kept brief. For further study the reader may be referred to [Def00b] and [JMP+99] and the DMSO website [Def00g].

3.1 Overview

The High Level Architecture (HLA) is a software architecture for distributed simulation. It can be used by developers to create simulation applications. When the HLA concept was developed, this was done before a background of a diverse environment that required extensive flexibility together with a minimum number of constraints. Military simulation applications cover a vast spectrum, from highly aggregated, discrete event simulators for the training of whole battle staffs to individual training simulators. And not only the training community relies on simulation, also the research and development communities have their own requirements for simulation applications.

So the HLA was developed to be flexible enough to cover this diverse set of simulation systems and sophisticated enough to embrace also the next generation of simulations [JMP+99]. The initial baseline definition of the HLA was done in 1995.

The HLA provides a technical framework to the developers that features a set of capabilities to support the design and execution of distributed simulations, composed of multiple simulation applications. In this context two key terms for dealing with the HLA are "**Federation**", for the integration of a set of interoperating simulation applications and "**Federate**" for the single simulation component that participates in a federation. A participant in a federation could also be an interface with a live system, like a flight simulator or a telemetry unit for experiments on a testing range.

In figure 3.1 we can see a functional view of the High Level Architecture. As a technical framework architecture, it enables the integration of simulation federates,

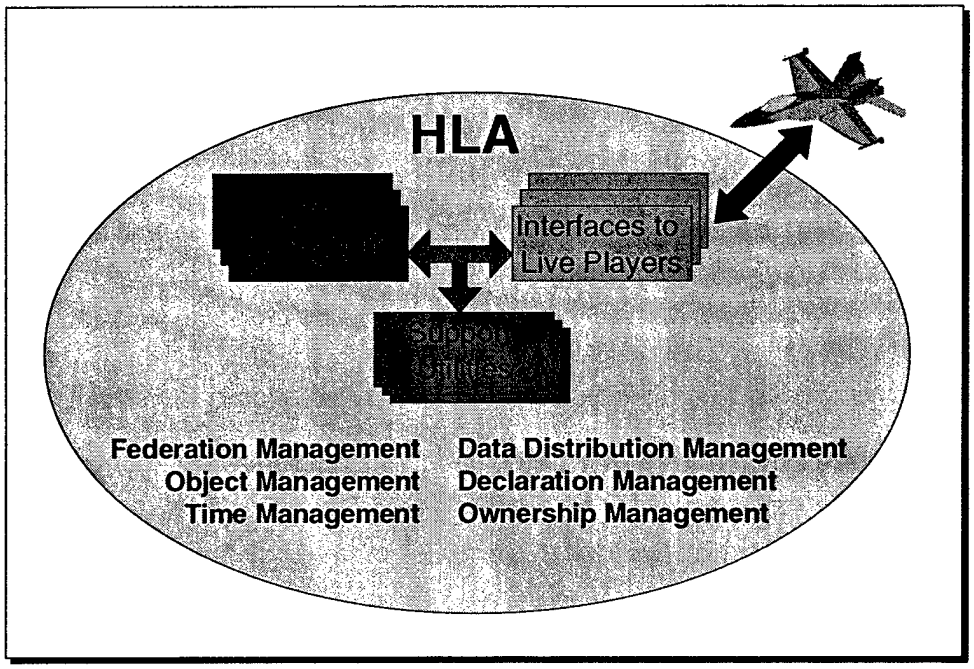

Figure 3.1: A functional view of the High Level Architecture

which can be genuine computer simulators, interfaces to "live" components and also support tools, into federations. This is done by providing definitions for:

- Federation Management
- Data Distribution Management
- Object Management
- Declaration Management
- Time Management
- Ownership Management

3.2 Simulation Interoperability and the Federation Development Process

When considering simulation interoperability, a distinction has to be made between two aspects of this field ([JMP+99]):

Technical interoperability refers to the issues related to actually making a federation of distributed and independent simulators work together.

Substantive interoperability addresses issues that impact on the ability of simulations to inter-operate in a coherent and logically meaningful way.

For example, the first item would include the definitions of inter-process and inter-platform communications, whereas the latter one would deal with questions like what data actually has to be exchanged between federates. Obviously, the two aspects are closely intertwined: Without interoperability on the technical level, the best substantive concepts cannot be put to work. At the same time, technical interoperability alone, without the substantive interoperability, is meaningless.

The HLA not only provides the developers community with the means to address the technical interoperability challenges, but also defines a process to organize and document the whole life cycle of the creation and execution of an HLA federation: The Federation Development and Execution Process, FEDEP. It comprises six steps:

Step 1: Define Federation Objectives

Step 2: Develop Federation Conceptual Model

Step 3: Design Federation

Step 4: Develop Federation

Step 5: Integrate and Test Federation

Step 6: Execute Federation and Test Results

3.3 The HLA Object Models

The architecture defines several object models, to standardize the development of HLA compliant software.

Federation Object Model (FOM): There is one FOM per federation and it introduces all shared information (i. e. Objects, Interactions). It also deals with inter-federate issues, e. g. data encoding schemes.

Simulation Object Model (SOM): The standard requires one SOM for each federate in the federation. It focuses on the federate's internal operation and describes salient characteristics.

Management Object Model (MOM): It defines all objects and interactions used to manage a federation.

Since especially the **Federation Object Model** will be of importance later in this work, we will go a little more into the details at this point. The first step to the implementation of HLA compliant software is the definition of the FOM. That means, everything that shall later be present in the "simulation space", has to be mapped

to the generic High Level Architecture constructs of Objects and Interactions. Both Objects and Interactions can have attributes to further refine the definition. Becauses an attribute to an Object can itself be an Object again, this leads to an object-oriented structure, where in the end everything is broken down to the level of generic simple data types (such as `String`, `int` or `byte`).

Both Objects and Interactions are of a similar nature. The difference is, that while Objects persist within the simulation space for a certain time (e. g. "Airplane", "Tank"), Interactions are instantaneous, non-persistent events (e. g. "Collision", "Radio Transmission").

The definition of the Federation Object Model is an enabling step, because it permits the sharing of information and the interaction between the federates, that will later be implemented. But it is also limiting, because only through the view defined by the FOM can federates "see" the simulation space and its other components.

To standardize the definition of the mentioned object models, the Object Model Template (OMT) exists. The OMT provides a common method for representing and documenting HLA object model information. Although the ideas of the Object Model definition process will be used all throughout this work, there was not no time to go through the detailed proceeding of using the OMT for definition and documentation.

3.4 High Level Architecture in real life: The Run Time Infrastructure RTI

The HLA serves as a theoretical framework, defining all necessary standards for the implementation of distributed inter-operating simulations. But there also exists an implementation that puts these concepts and interface specifications to work and provides common services to simulation systems –the Run Time Infrastructure, RTI.

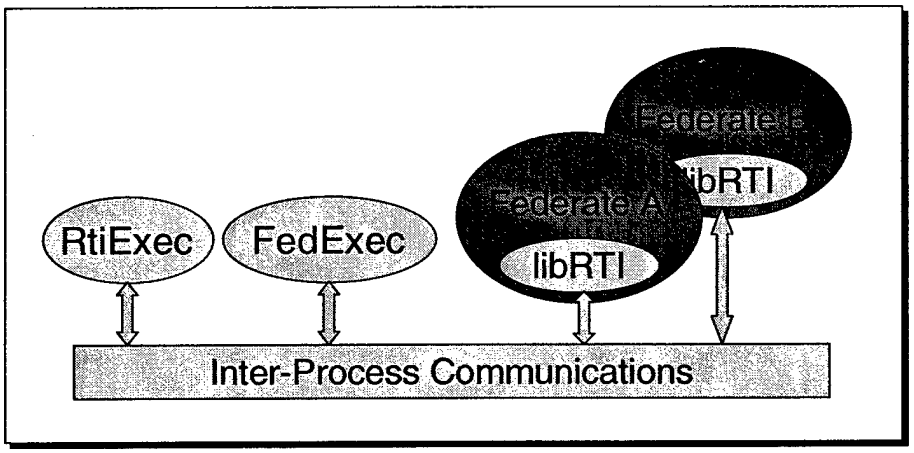

Figure 3.2: Functional view of the Run Time Infrastructure – RTI

The RTI is provided to the developer as a software package that contains all necessary components to implement an HLA compliant simulation application and then to run whole simulation federations. The main components of the software package are:

- Class libraries that allow the developer to access the RTI services from his programs.
- Core system programs, through which all RTI inter-process communications

and interactions are handled. Concretely, these are the RTI background process `rtiexec` and the federation execution background process `fedex`.

Figure 3.2 shows, how these components work together: Linked by inter-process communications, we have several federates who access the RTI services by using the RTI programming interface[1]. The figure reveals nothing about a possible spatial distribution, though. All the depicted components could be on different computers and operating systems. In that case, everything is interconnected by the local are network (LAN). Internally the RTI relies on TCP/IP for communications.

One of the central goals of the HLA, **simulation interoperability**, is ensured on a technical level by providing **platform interoperability** through the RTI. Currently, the runtime software is available for eleven different operating systems (e. g. Sun Solaris, Windows95, 98 and NT, Irix, Linux,...) and a variety of programming languages (C++, Java, Corba,...). A simulation application that is built on one operating system/programming language pair is technically able to join any federation, no matter which systems and platforms the other components rely on.

All RTI software, documentation and even supporting tools, is available from the Defense Modeling and Simulation Office (DMSO). It can be downloaded via the internet from their download server [Def00a]. But because this is military material, although not classified, some restrictions apply: Before being able to download and use HLA related material, the developer has to apply for a user account. Also, the RTI must not be exported outside the United States or given to anybody else by the approved user.

[1] denoted by "libRTI" in the graph

This work is based on the "RTI Next Generation 1.3 Version 3.1", which was effective during the implementation phase. In the meantime, the version has been advanced to 3.2.

3.5 The RTI and Timing

In the previous chapter, different concepts for simulation timing have been introduced. The Run Time Infrastructure can handle all these different concepts, which proves its versatility. Federations that do not deal with time at all can be implemented just as well as federations using highly sophisticated real-time synchronization schemes.

In general it can be stated, that each federate has its own understanding of time. There is no such thing as an universal, common simulation time. But the RTI provides means for the federates to coordinate their personal perception of time within the federation, based upon a scheme that has been agreed upon.

To implement the timing behaviour, federates can set two attributes, and by doing this, declare their own intentions and significance regarding the overall federation timing:

- `timeConstrained`: This only means, that the federate is bound to federation timing.
- `timeRegulating`: By declaring this, the federate states that other time constrained federates will be bound to this federate's local time.

This allows for many possible combinations. For example, all federates could be

constrained and one of them additionally regulating, to implement something like a "Master-Slave" concept.

During the simulation, federates make known their intention of advancing their local time by posting a `timeAdvanceRequest` to the RTI services. According to the overall timing situation (i. e. how the before mentioned attributes are set and the other federate's times throughout the federation), the request ist then granted by the RTI, as soon as it becomes feasible.

3.6 Applications for the RTI

Statistics of RTI downloads and of DMSO user accounts show that the RTI software is already being actively used by the military simulation community. Besides the HLA's formidable capabilities, one reason for this is most likely the Department of Defense policy to require all military simulations to be HLA-compliant by beginning of the year 2001. Although not everywhere could this deadline be met, it still makes clear the DoD's strong commitment to the HLA as the future foundation for all its simulation efforts.

As part of this work an attempt was made, to find out more about which simulation projects already use the HLA, to what extent and with how much success. However, with the author being a civilian, no detailed information about projects utilizing the HLA could be obtained. It seems though, that the efforts to incorporate the HLA are most advanced in the military training community. Different training simulators are interconnected to form whole battlespace training situations. Also the Joint

Warfighter Program builds on HLA compliance for its simulation efforts.

Now the applicability of distributed simulation and the HLA for CMIF's project partner, the Edwards AFB Range Testing Facility, has to be considered. Recent studies at CMIF proposed a distributed simulation environment as a necessary enhancement for EW range testing. The reason for this is, that the testing facility will soon be faced with testing aircraft-type systems whose basic employment concepts have them dependent on other system components. This means, that the airborne platforms, usually the actual hardware test-objects, are not really independent, but a part in a complex network.

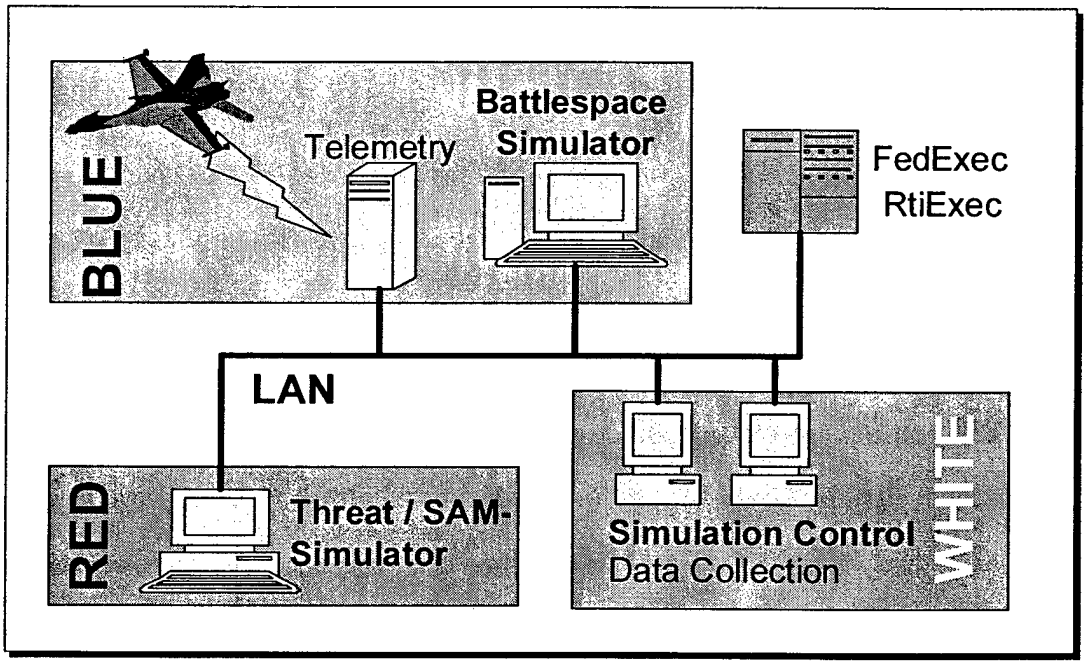

Figure 3.3: This scenario shows how in the future range testing could be enhanced by combining real (flying) components with virtual ones in one "simulation space".

Up to now, the range testing at Edwards is done in a one-on-one situation, where the to be tested EW component is subjected to stimulation by a single threat, a

hostile SAM system for example. This proceeding neglects the fact that today's battle scenarios resemble a network of interacting nodes, like fighters, AWACS radar surveillance airplanes, command and control posts, etc. And the same is the case for the enemy side as well.

Obviously it is not feasible to test a collection of full-scale system components. Thus driven by affordability concerns, a need for distributed simulation arises. Previous CMIF research proposes a Test and Evaluation (T&E) infrastructure based on a distributed simulation approach.

Figure 3.3 illustrates a possible future scenario for range testing, where the to be tested hardware is surrounded by an environment that is partly virtual and only partly real[2].

It must be understood though, that the proposed distributed, part-virtual environment will not be able to deal with the signal layer of the involved EW components. The virtual sections of the environment would in some way be limited to a higher level, and not deal with the very electromagnetic waves and signals, which are at the core of EW component testing. Simulations, which communicate themselves by means of electrical signals, would simply be too slow to emulate other high-frequency electrical signals. But because the same limitations exist for the connections between the real operational platforms, the simulated environment will still be able to contribute to the experiment and not be too far from reality.

[2]In this context, the following color coding is common in the military: "Blue" depicts the friendly side, whereas "Red" stands for foe. "White" denotes the environment perspective, meaning neutral observers, controllers and eventual supporting services.

Before this vague background it is rather difficult to make definitive statements about the applicability of the HLA for Edwards, or about the expenditure necessary to implement the proposed T&E Framework using HLA. So this section can rather be understood as a motivational scenario than as a technical concept.

Distributed simulation is not yet a fully mature science. There are many research issues to be considered in the course of defining the proposed T&E framework and the corresponding details before Edwards AFB Range Testing Facility will have sufficient knowledge and confidence to decide on an approach to implementing the framework. Thus there is a clear role for related univerity-type research to support Edwards in defining and designing the eventual framework approach to a new T&E infrastructure.

Chapter 4

The CMIF HLA Environment

4.1 Purpose of the CMIF HLA Environment

The Center for Multisource Information Fusion (CMIF) conducts research in various fields of distributed data fusion, taget tracking and also multi-agent systems. A significant part of this work is usually done by single student or small student teams. Naturally, the personal committment of an university student ranges rather over a couple of months than over several years, as it would be the case in a professional environment.

It would be desirable to utilize "real" distributed simulation, with all its before mentioned benefits, for the CMIF projects where applicable. But at the same time, the manpower available for these projects should not be wasted on familiarization with details of the RTI concept and implementation. Instead, the focus has to remain on distributed data fusion research.

The purpose of the CMIF HLA Environment is to bridge the gap resulting from the two goals introduced in the last paragraph: Make the RTI available to CMIF's research projects whithout requiring in-depth RTI knowledge or programming experience. Also, a reusable tool for simulation support is desirable, enhancing the simulation testbed with functionalities for simulation configuration, control and evaluation.

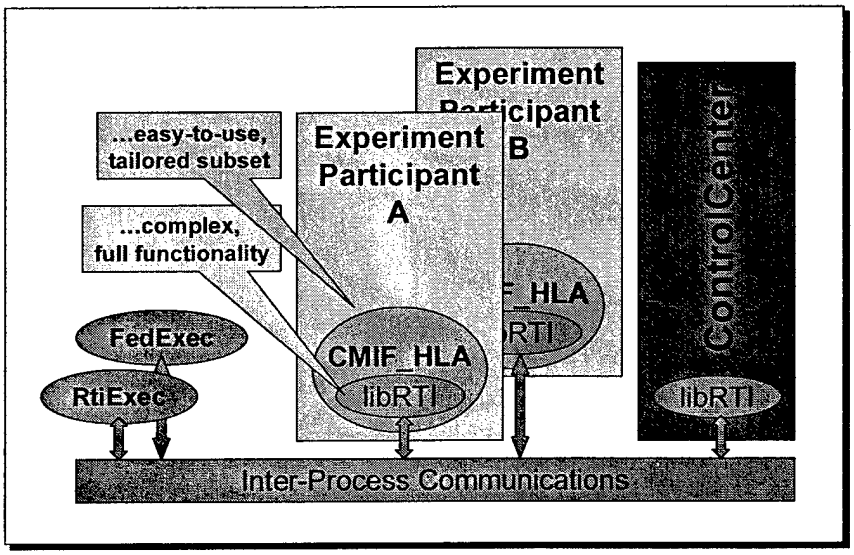

Figure 4.1: This graph shows a conceptual view of the CMIF HLA Environment. The complex RTI is masked from the developer of experiment participants by the CMIF programming toolkit, whose tailored functions are easier to use.

Figure 4.1 shows the concept of the CMIF HLA Environment. It is to be understood as a further development of the functional view to the Run Time Infrastructure from figure 3.2. Again we have several federates hooked up to the RTI services and thus populating one HLA defined "simulation space". But this time, the developers of the federates "Experiment Participant A" and "Experiment Participant B" do not

have to deal with the RTI Application Programming Interface (API)[1] but instead use the API provided by the CMIF HLA Environment. The CMIF API will ensure the following:

- The complex functionalities of the RTI are hidden from the application developer.
- Simple means for communication between experiment participants are made available.
- The functionalities of the supporting tool "control center" are accessible by a few simple method calls.

For technical reasons, a distributed simulation can never start right away with the pursuing of its simulation goal. Issues like establishing contact between the components and negotiation of key simualation parameters have to be taken care of first. In order to better organize this, the following set of six "Experiment Phases" was defined for the CMIF HLA Environment. Each phase requires a specific behaviour for the experiment participants as well as for the control center.

Phase I **Configuration:** This phase covers all steps necessary to configure an experiment within the CMIF HLA Environment. Defining which experiment participants will be present and setting timing parameters, for example.

Phase II **RTI Setup:** Initiation of contact to the RTI services, (in case of the control center:) launching the federation execution and establish contact between control center and experiment participants.

[1] denoted by "libRTI" in respective graphs

Phase III **Simulation:** The experiment participants perform their actual simulation work, the control center provides the timing and other support services.

Phase IV **RTI Cleanup:** Orderly termination of contact to other federates and shutdown of the federation execution.

Phase V **Shutdown:** Cleanup and termination of the program itself.

Another phase could be "Simulation Postprocessing", following after the "Simulation" phase. It would cover everything related to the processing and evaluation of experiment results. Because no such evaluation functionality is implemented in the support tool "control center" so far, this phase was left out. An augmentation might later be necessary.

4.2 The Federation Object Model

This section refers to the definition of HLA object models as introduced in section 3.3. The Federation Object Model (FOM) contains the object-oriented definition of Objects and Interactions, representing the common view that all federates have of the simulation space.

To implement the CMIF HLA Environment, only two kinds of objects are necessary. They will be introduced from a FOM point of view at this point, but more detailed information will follow later in the according sections.

CMIFControlCenter

The control center will be the central federate within the CMIF HLA Environment,

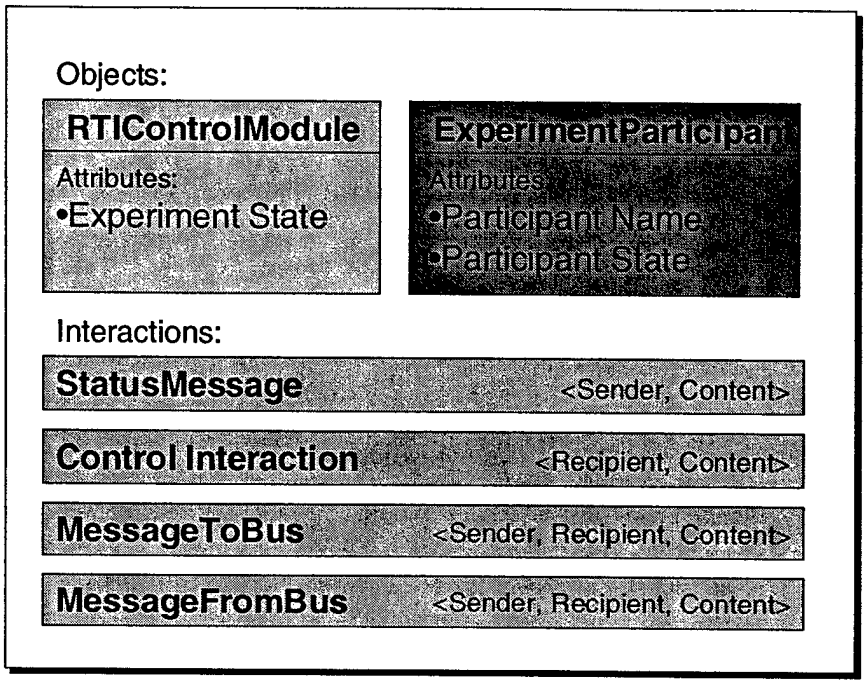

Figure 4.2: This graph shows the Federation Object Model for the CMIF HLA Environment with all objects and interactions, including their attributes.

responsible for all coordination and control issues. In the simulation space, it will be represented by the HLA object `RTIControlModule` (the nomenclature will become more obvious once the actual implementation of the control center and its class structure will be discussed). The only attribute to the control center will be `ExperimentState`, containing coded status information.

CMIFExperimentParticipant

Because the CMIF HLA Environment is again, very much like the High Level Architecture itself, a neutral testbed, it is not possible and desired to predefine what later participating federates in this environment will be like or what they will do. The only thing that is for sure is that they will interact with the control center and, of course, with each other. The first aspect refers to the environment's administrative

functionalities, whereas the latter one refers to the actual experiment content, the simulation specific behaviour.

Based on this, we introduce as an HLA object the generic `CMIFExperimentParticipant`. The only attributes neccessary for the abstract experiment participant are `ParticipantName`, for identification, and `ParticipantState`, which will contain status and "health" information.

Also the following set of **interactions**, meaning: non-persistent objects, are defined for the CMIF HLA Environment:

- `ControlInteraction`: A means for the control center to control experiment participants. The two attributes are `Content` and `Recipient`.
- `StatusMessage`: This interaction lets the experiment participants send administrative information back to the control center. Attributes are `Content` and `Sender`.
- `MessageToBus`: The "Bus" represents the virtual communication line for inter-experiment participant communications, which are in reality routed through the control center. `MessageToBus` is a message issued by `Sender` for the `Recipient`.
- `MessageFromBus`: This is a `MessageToBus` after processing and rerouting by the control center.

All listed interaction attributes are of type `String`, whereas the status attributes of the two objects are of type `int`.

The definition of the Federation Object Model is the binding step, and since the RTI

relies on *late-binding*[2], all this is done by means of a federation configuration file that is interpreted at runtime. More precisely: at the initiation of the federation execution background process `fedex`. The file `CMIF_HLA_Environment.fed`, which defines the FOM for the CMIF HLA Environment, can be found in appendix B. While the `.fed` file also contains many higher-level parameters, the code that defines our specific objects and interactions can be found in two relatively small sections that are on pages 81 and 84. An overview to the whole CMIF HLA Environment FOM is given in figure 4.2. A more detailed description of the two major objects will be provided in the subsequent sections.

4.3 The CMIF Control Center

4.3.1 Overview

The CMIF control center is supposed to serve as a "White" control tool for experiments whithin the CMIF HLA Environment. Its main functions are to provide the user with control over and information about:

- experiment configuration,
- the simulation run,
- experiment participants present in the simulation space and

[2]This means, the definition of the object-structure is not hard-coded into the RTI software. While increasing the complexity, this is also the prerequisite for the RTI's high flexibility. The opposite concept would be *early-binding*.

- data collection for evaluation.

You could also think of the control center as an "operator's desk" from where the simulation environment can be supervised. Some functionalities that are already integral parts of the CMIF HLA Environment concept, could not be implemented yet. Mainly these are: Data collection, evaluation of the amount of data being transferred between participants, emulation of communication "bandwidth" and advanced message routing. The implementation of these functions will follow with the next version of the CMIF HLA Environment package.

4.3.2 Class Structure

Because the implementation of the control center was a major part of this work, a few more details about its inner concept will be provided at this point. The control center's functionalities are distributed over the following classes:

- `CMIFControlCenter`: The main class, linking all other components. It is responsible for startup of the program and initialization of all major components.
- `CMIFControlDisplay`: This class bundles all user interface functionalities. Display components are initialized and maintained.
- `CMIFExperimentManager`: The experiment manager is active during the configuration phase and handles all related tasks.
- `RTIControlModule`: This class is responsible for all activities related to the RTI. It establishes and maintains contact to all background services and to the

experiment participants during simulation setup and run.

- `RTIControlModuleFedamb`: The final interface class, linking the `RTIControlModule` to RTI services.

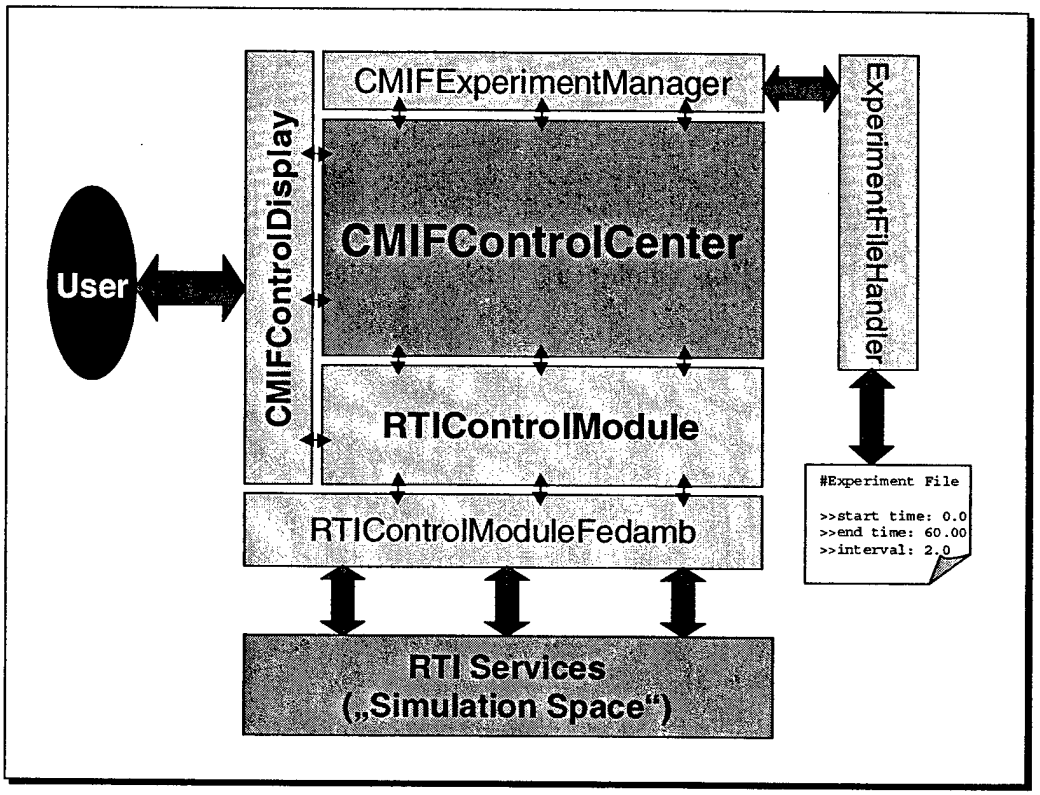

Figure 4.3: Overview over the control center class structure and interactions with outside components.

Also, some utility classes were developed, for example the classes `DebugHelper` and `ExperimentFileHandler`. The first one makes the output of debugging information (during development) easy and controllable. The latter one reads and writes the `.cef` experiment configuration files. See figure 4.3 for a depiction. The whole sourcecode can be found in the appendix, section D.1.

4.3.3 Graphical User Interface for the Control Center

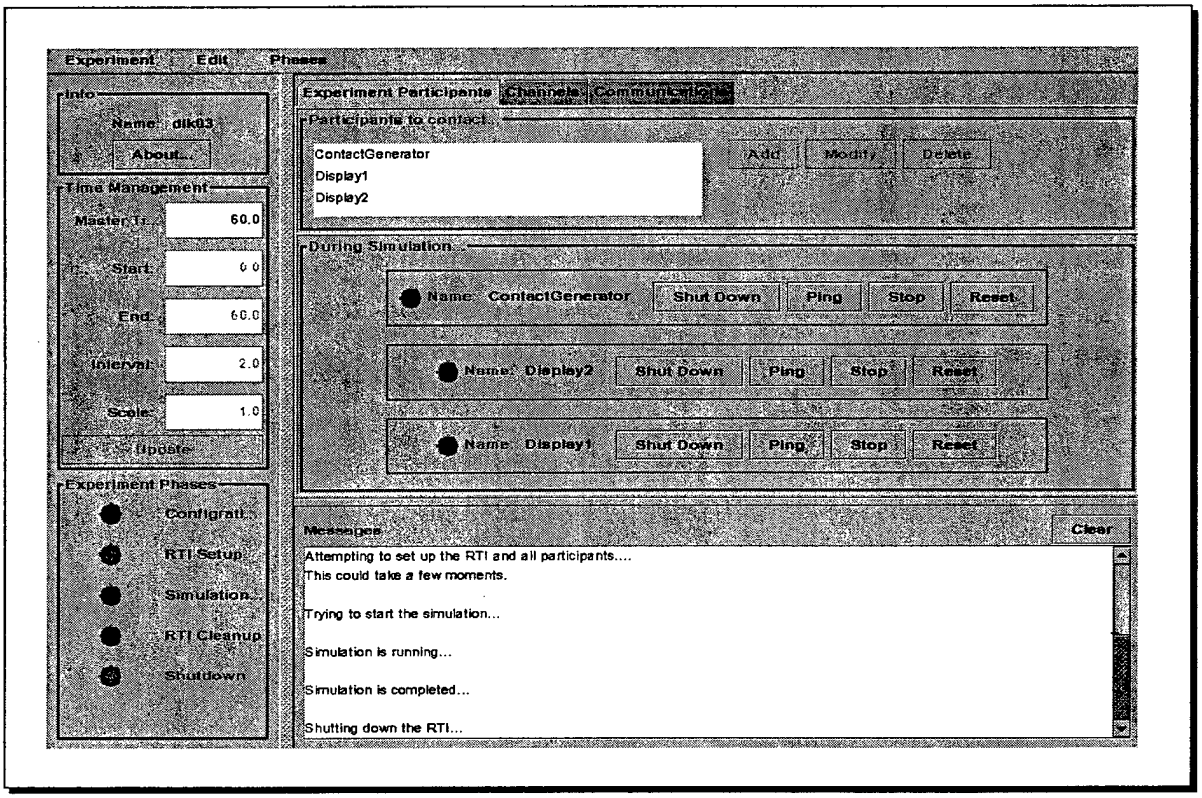

Figure 4.4: Screenshot of the complete control center user interface

Figure 4.4 shows a screenshot of the control center's graphical user interface (GUI). In the following, the main sections and their functionalities will be introduced.

Menus

In the current version, the control center contains two main menus, `Experiment` and `Phases`, offering the following functionalities: The `Experiment` menu contains items for handling the overall experiment configuration and the configuration files:

- New
- Open Old Experiment

- Close Current Experiment
- Save Configuration
- Save As...
- Exit

These should be more or less self-explanatory. `Save...` writes the `.cef` file and `Exit` terminates the whole control center application. The `Phases` menu lets the user switch between the before mentioned "Experiment Phases":

- Configure Experiment
- Setup RTI Execution
- Run Simulation
- Abort Simulation
- Clean Up RTI

A screenshot of both menus can be seen in figure 4.5

Info Panel

This panel simply shows the name of the current experiment configuration and allows to access additional information that has been stored about this experiment (see figure 4.6).

Timing Panel

This section of the graphical user interface allows to configure the timing parameters

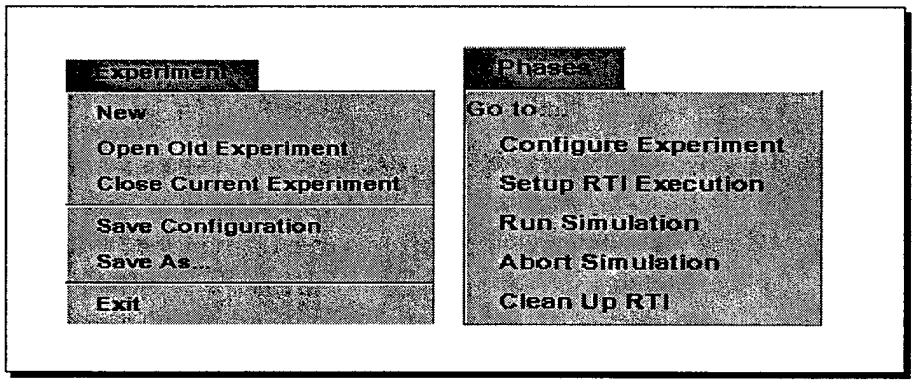

Figure 4.5: The control center `Experiment` and `Phases` menus

for the current experiment. On top, the master time, which progresses from start to end time during the simulation run, is displayed. The key values are: `Start`, `End`, `Interval` and `Scale`. The first two define the section of time that the simulation run will cover and the third sets the length of the time steps along which the time progresses. All three represent seconds. The last value defines the scaling factor which determines, how real-time and experiment time are related to each other (ref. figure 2.2). Changes in the timing section can only be made during the configuration phase and not while the RTI is being set up or the simulation is running (see figure 4.6). To make any changes effective, the `Update` button has to be pressed first.

Phases Panel

As a simple means for the user to see what state the control center is in at a given moment and what steps can/must be taken next, this panel shows the experiment phases together with colored icons. The icons have the following meanings:

- Green, blinking: This phase is currently in progress,
- Green, steady: This phase has successfully been completed,

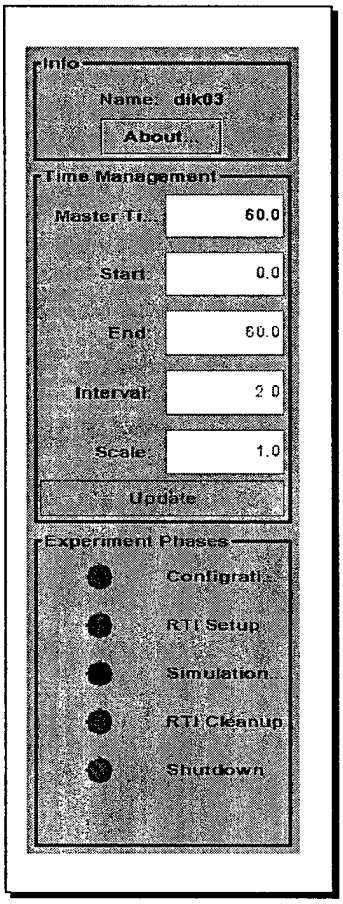

Figure 4.6: This screenshot shows a close-up of the left side of the control center's user interface. It contains sections for experiment information, timing parameters and the simulation phases.

- Orange: It is possible to switch to this phase,
- Red: Switching to this phase is not possible.

Experiment Participants

Of the three panels situated on the upper right side of the main window, only one contains actual functionalities in the current version of the control center. The "Experiment Participants" panel has an important role in the control center's concept:

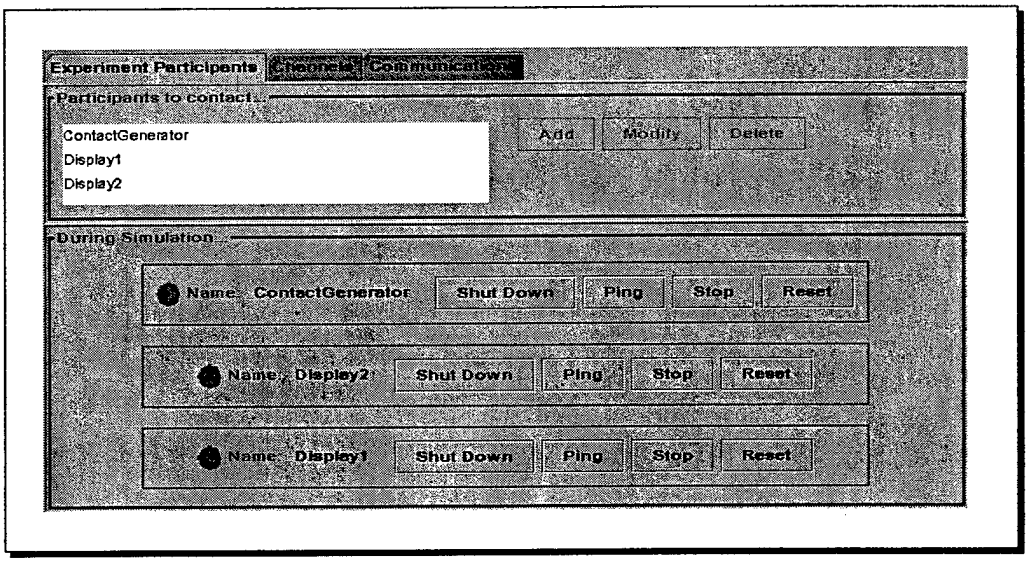

Figure 4.7: This section of the control center GUI informs about the experiment participants. The upper part represents the list of participants to contact, below is the dynamic representation of experiment participants during runtime.

Because the control center operates independently from experiment participant applications, the participants have to be contacted during runtime. Also, the control center does not "know" about what kind of applications will be present in the simulation space. So the user has to provide the control center with a list of all experiment participants that are to be expected. This is done in the upper part, where, during configuration, the user can add, remove and modify names on the "Participants to contact..." list.

The lower section ("During Simulation...") is a dynamic representation of the control center's perception of the simulation space. For every experiment participant that is discovered during the RTI setup phase, an information and control panel is created. It displays the experiment participant's name, the current status (colored icon) and allows remote control via control interactions. These are triggered by using

the `Shut Down`, `Ping`, `Stop` and `Reset` buttons. The first two do work, the latter two have no effect in the current version of the software.

Only after the number of participants and their names in the lower, dynamic, list match the entries of the configuration list above does the control center declare the RTI setup phase for completed. See also figure 4.7.

Message Area

This area on the lower right side of the control center's window is used to display messages and feedback–information, errors, status, etc. – to the user.

4.4 The Experiment Participant

The other object that is defined by the CMIF HLA Environment federation object model is the `CMIFExperimentParticipant`. It is a generic representation of an arbitrary application that will in some way be involved in a distributed simulation experiment. This high level of abstraction makes it difficult to describe what the `CMIFExperimentParticipant` actually is, but reflects the very underlying concept: Think of what a distributed simulation with its components could be like and try to boil down their behavoir to a very general level. This leads to the statement, that all that any participant in a distributed simulation does, is the following:

- perform specific work, related to the simulation experiment,
- communicate with the other simulation participants as part of the simulated behaviour and

- communicate with the infrastructural services, support tools etc.

This can be interpreted as two separate levels: The *Simulation Level* deals whith the actual purpose of the experiment, e. g. the tracking algorithms in a tracker component. Below that, on the *Administrative Level*, all the necessary tasks can be found that make the application actually participate in the distributed environment, negotiating timing and synchronization for example.

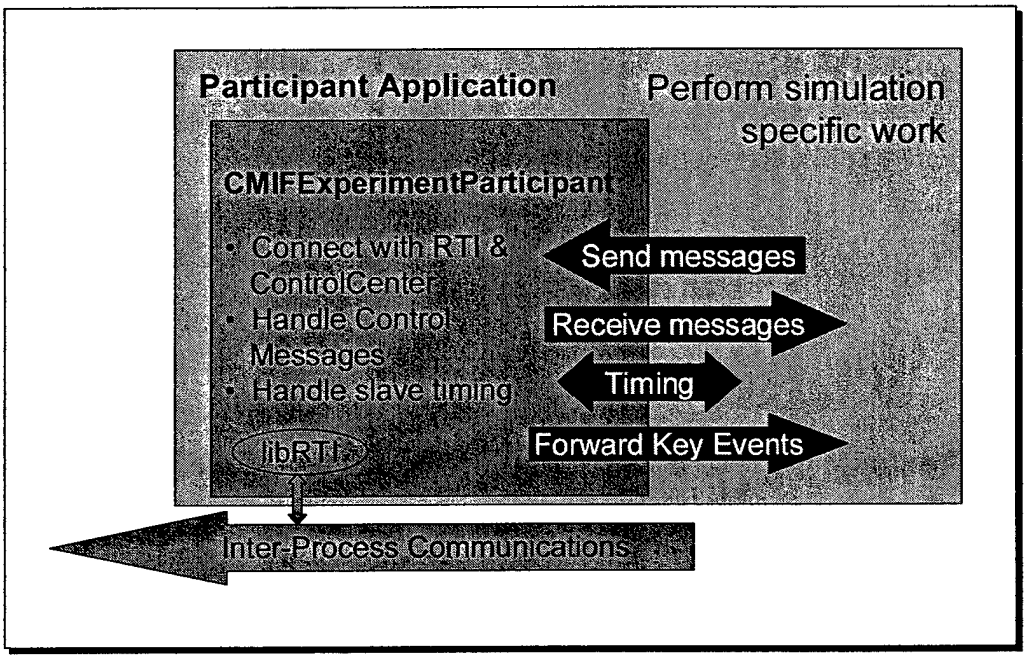

Figure 4.8: An overview over the experiment participant concept

Figure 4.8 visualizes how this concept is put into action: Any experiment participant application will subclass the generic `CMIFExperimentParticpant` and thus inherit its capabilities of dealing with the RTI services and communicating with the control center, the *Administrative Level* functionalities. Now the developer has to implement the simulation-specific behaviour to make his experiment participant actually do some-

thing. He also has to define the inter-participant communications, but does not have to go all the way down and work on the *Administrative Level* in order to do so. Instead, easy means for every contact with the "outside world" are provided by the `CMIFControlCenter` superclass.

From an HLA point of view, which brings us down to the *Administrative Level*, the experiment participant issues `MessageToBus` and `StatusMessage` interactions and itself is recipient to `Control-` and `MessageFromBus` interactions.

4.5 Implementation Remarks

For the implementation of the CMIF HLA Environment, the Java programming language was chosen. The main reason for this choice was the platform independency that Java provides. Code that is developed and compiled on one operating system will work on most other systems too (with some minor technical limitations, of course). With the CMIF lab's computer network being a conglomeration of various operating systems and hardware platforms, using Java seemed like the only feasible approach for providing the lab with a usable software package for distributed simulation. Also while Java features all aspects of a high-level object oriented computer language, it is less error-prone (i. e. programmer's errors) and is generally very suitable for prototype development. Last not least, with Java's fuctionalities it is especially easy to develop appealing graphical user interfaces. A prerequisite was of course, that the run time infrastructure is available in a Java version, too.

The choice of Java also has some drawbacks:

- Subsequent development of applications will be limited to Java [3]
- Java is not an ideal platform for the development of mathematical and numerical software and lacks a lot of functions in this regard. Add-on packages might be available, though
- The advantage of platform independency comes at the price of slower performance. Although this is a general problem of Java, no significant problems could be observed at the current stage of development of the CMIF HLA Environment.

[3]Theoretically, any other RTI supported language would be fine too, but that would require to start over at a deeper level and thus an in-depth knowledge of how the RTI and the CMIF HLA Environment work. Thereby, the initial goal of simplicity would be foiled.

Chapter 5

A Programmer's Guide: How to Install and Use the CMIF HLA Environment

This chapter is supposed to serve as a tutorial and programmer's guide for the implementation of distributed simulation experiments within the CMIF HLA Environment. It will cover all steps necessary, from installation of the software infrastructure over programming advice up to the actual execution of a simulation run. The latter two steps will be illustrated by guiding the user through the development of a sample program, the `DemoApplication`, which is part of the CMIF HLA package. Prerequisites for planning and implementing an experiment are knowledge of the Java programming language and familiarity with the specific operating system(s) for the installation process, because the RTI as a developer's tool is quite complex.

Please note that there will be no introduction on how to directly access the RTI within own code, because the whole purpose of the CMIF HLA Environment is to make this step obsolete for new users. Those who want to get into this highly complex area may be referred to the detailed doucmentation that comes with the RTI software ([Def00b] to [Def00d]). Also studying the commented source code of this work (see appendix D) or sample programs provided with the RTI can be extremely helpful.

5.1 Download and Installation of the RTI

Before using any part of the CMIF HLA Environment, a functioning installation of the Run Time Infrastructure is needed on all platforms and computers that will be used in an experiment setup. Because a very detailed description of the process, also addressing the different platforms, can be found in the "RTI Installation Guide" ([Def00f]), the following list covers the proceeding only in brief:

Download the Run Time Infrastructure software from the DMSO server at `http://sdc.dmso.mil`. This requires to obtain a user account first. The software is provided for different platforms and programming languages from which the user can choose the appropriate configuration. There usually are installation prerequisites, like the presence of certain C-libraries or C-compilers on the system, but these prerequisites are well documented.

Note that the RTI implementations in programming languages other than C++ are usually add-ons to the C++ core system. For example, for the CMIF HLA Environment (which uses Java), the C++ as well as the Java package have to

be downloaded and installed. The specific packages used for this work were:

`RTI-1.3NGv3.1-SunOS-5.6-Sparc-SPRO-4.2-opt-mt.sh` and
`RTI-1.3NGv3.1-SunOS-5.6-Sparc-SPRO-4.2-opt-mt-JavaBinding.sh`.

It can be seen, that even the name of the download file reveals the key information about the specific configuration.

Unpack the software to a suitable directory, the core and the Java packages to the same one. The RTI packages usually come as self extracting executables (`.exe` and `.sh`) or have to be unpacked using the `ZIP` (Windows) or the `gunzip` and `tar` (UNIX) utilities. The resulting directory structure after unpacking is standardized for all RTI versions and can be seen in figure 5.2.

```
...
#RTI variables:
export RTI_HOME=/home/kharth/HLA/RTI-1.3NGv3.1
export RTI_BUILD_TYPE=SunOS-5.6-Sparc-SPRO-4.2-opt-mt

export LD_LIBRARY_PATH=
        $LD_LIBRARY_PATH:$RTI_HOME/$RTI_BUILD_TYPE/lib

#RTI Java
export JAVA_BINDING_HOME=
        $RTI_HOME/$RTI_BUILD_TYPE/apps/javaBinding

export CLASSPATH=$CLASSPATH$JAVA_BINDING_HOME/classes:.
export LD_LIBRARY_PATH=
        $LD_LIBRARY_PATH:$JAVA_BINDING_HOME/lib
...
```

Figure 5.1: Sample configuration file `.bashrc`

Adjust the environment variables. The RTI relies on this mechanism to facilitate the management of different RTI versions or configurations in one filesystem. The variables point to the directories where the system files are to be found and the following settings are needed:

- `RTI_HOME` – set to the directory where the RTI is installed
- `RTI_BUILD_TYPE` – set to the installation specific build type
- `JAVA_BINDING_HOME` – directory where the RTI Java classes can be found
- `LD_LIBRARY_PATH` – path to shared object libraries (UNIX specific), both C++ and Java
- `CLASSPATH` – path to Java classes

Under UNIX these settings are usually defined in the `.bashrc` or `.aliases` configuration files. See figure 5.1 for a sample section of such a file. On Windows platforms the approach is slightly different, please refer to [Def00f].

Modify the configuration file `RTI.rid`. This file contains important system and networking settings for the RTI. For now, one parameter has to be defined: Open the file in a text editor and change the setting for `RtiExecutiveEndpoint` to the hostname (and port number) of the computer that will be running the RTI background process later. The port number is a five digit number that is arbitrary as long as it does not interfere with any other service that is using the network. If all components of the experiment will be running in different processes on the the same computer, the line can be commented out all together. See appendix A, line 4, for a sample setting. You can keep several `RTI.rid`

files with different settings. When, during runtime, the federation execution is created (`fedex` process), the program always looks for the file automatically in the same location from where the control center was launched.

5.2 Installation of the CMIF HLA Environment

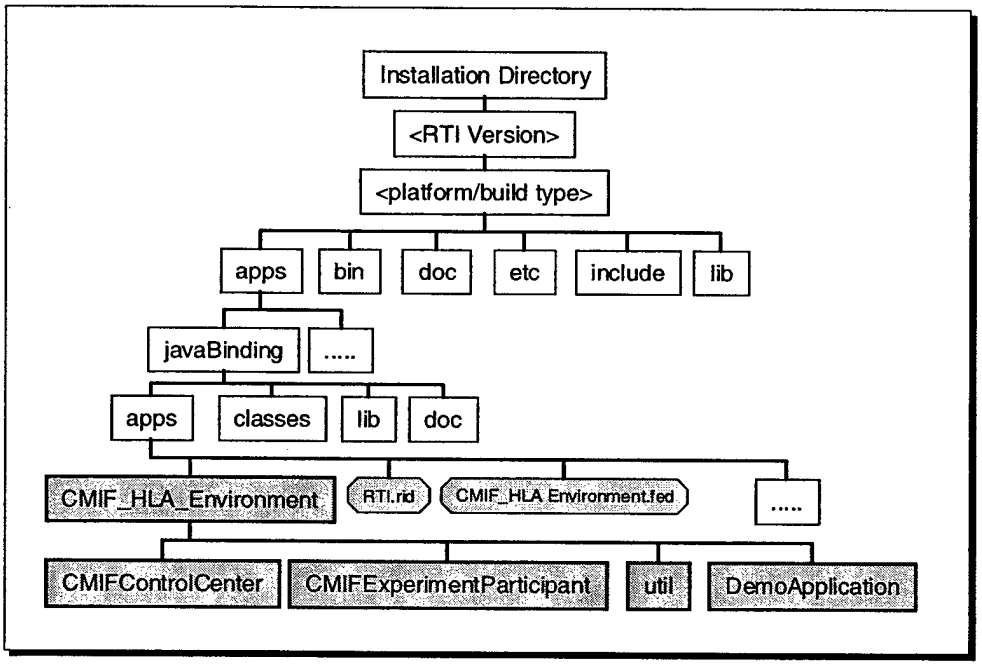

Figure 5.2: This graph gives an overview over the directory structure of the CMIF HLA Environment. White boxes indicate a common RTI installation, grey indicates parts of the CMIF HLA Environment.

The latest version of the CMIF HLA Environment can be obtained from the author. You will receive either a `.zip` (Windows) or `.tar.gz` (UNIX) file which has to be unpacked into the directory `$JAVA_BINDING_HOME/apps/`. The resulting directory structure, which is added onto the common RTI installation, can be seen in figure 5.2. The directory `CMIFControlCenter` contains all the classes that belong to the Control

Center whereas `CMIFExperimentParticipant` contains all classes of the generic experiment participant. Parallel to the `CMIF_HLA_Environment` directory, the two RTI configuration files can be found: `RTI.rid`, defining networking and system parameters, and `CMIF_HLA_Environment.fed`, the file defining the federation, its objects, interactions and attributes (see appendices A and B). This also is the point from where CMIF HLA related applications have to be started.

The current version of the CMIF HLA Environment has been developed and tested under Sun Solaris 2.6, using the "RTI-1.3 Next Generation Version 3.1". Also a test under Linux (Debian 2.2) has been successful. Normally the software should also work on other platforms or with newer versions of the RTI, but limitations may exist.

For the Java programming, a Java Software Development Kit (SDK) for Java 2 is needed which supports Java native threads. For details see the Sun Microsystems website at `http://www.sun.com`.

5.3 Implementing an Application

What follows is the description how to actually implement own experiment participants or "agents" that can form an experiment within the CMIF HLA Environment. This is the point where the transition from the abstract *early-binding* stage to *late-binding* is made and the generic experiment participant is developed into an application with specialized behaviour.

The main step to make all services accessible to an application is to subclass the generic `CMIFExperimentParticipant`. This class masks all low level communication

with the RTI from the developer and provides two-way interaction with the rest of the system by means of the inheritance principle: The application can act by calling methods provided by and inherited from its superclass. At the same time it can react through the callback concept. "Callback" means, that during program execution, the super class will invoke methods that are empty on the superclasses level itself. Only if they are overridden in the subclass and given any specialized behaviour, something actually happens.

5.3.1 Programming Toolkit Overview

5.3.1.1 Setup

In this case, "setup" means the initialization of the contact to the Run Time Infrastructure and, after that has been accomplished, the contact to the control center. The key steps are as follows:

1. Establish contact to and join the `CMIF_HLA_Environment` federation execution. This of course means, that it must already exist. The federation execution is created by the control center, during the RTI setup phase.

2. Report to the control center and exchange status information.

3. Receive key simulation information (i. e. timing) from the control center.

These tasks are processed in the background. All that has to be done at the application level to get started is to call the method

`doSetup()`.

This does not necessarily have to be at the beginning of the program, just prior to any RTI related activity. The setup process is executed asynchronously in its own thread, which means that the call to `doSetup()` returns almost immediately and other things can be done. The applicataion is then notified of the succesful completion of all setup tasks by a callback to

`doneWithSetup()`.

Furthermore, a callback to

`timingParametersChanged()`

indicates, that the timing parameters (i. e. start and end time, increment and scale-factor) have been set after negotiation with the control center. After these steps, the application has nothing to do except to wait for the signal from the control center that the simulation run has started. This is accomplished by calling the method

`waitForSimulation()`,

which puts the execution on hold until the next phase becomes active.

5.3.1.2 Simulation Run

As discussed earlier, there are two major paradigms for federate behaviour: Time-Stepped and Event-Based.

A Time-Stepped application will follow the simulation timeline and somehow determine its actions based on the current time. To implement this behaviour, the following concept is needed: After simulation start, the program has to enter a `while`-loop. At one time within the loop, a call to the method

`advanceOneStep()`

has to be made to force the federate to the next step in time (old time + increment). Alternatively, a call to

```
advanceTo(double newTime)
```

overrides the time increment parameter and attempts to jump directly to `newTime`. The call to one of these methods does not return until the desired new time is feasible, which means synchronous to the control center's master time. As a condition for the `while`-loop,

```
boolean simIsRunning()
```

can be used, which returns true or false to indicate if the simulation is still in progress. Furthermore, with

```
double getTime()
```

the current federate time can always easily be obtained.
The whole thing will look like this:

```
while ( simIsRunning() ) {
  //do something...
  //...and something else

  advanceOneStep();
}
```

An experiment participant with Event-Based behaviour is much easier to implement. Nothing special has to be done. The processing of incoming or outgoing messages and interactions is already ensured by an independent background thread. Any action will really be a reaction to some kind of input, live user/"player" or message. The means and methods for processing incoming interactions to the application will be discussed in the next section. Time, in this scenario is disregarded.

The most likely and realistic case however will be a combination of both Time-

Stepped and Event-Based behaviour, which requires a very careful aproach. All situations where one part could block out the other one have to be strictly avoided. For example, the program might be lying dormant to wait for user input and, while doing so, not perform the time-stepped tasks. Other available methods and callbacks related to the simulation run are

- `doneWithSimulation()`: Another notification that the simulation run is over. This can be used in Event-Based experiment participants, because these would not have the simulation `while`-loop.
- `updateSimTime(double simTime)`: Whenever the simulation time has successfully been changed to a new value, this callback method is activated.
- `abortSimulation()`: Notification that for some reason the simulation has been aborted.

5.3.1.3 Communications

This section deals with the methods and callbacks that are involved when communicating with the "outside world", i. e. other experiment participants or the control center. All data exchange is done with `String` data types. For details how to convert other data types to and from `String` representation, please refer to the Java API Documentation [Jav00b], classes `String`, `StringBuffer`, `Integer`, `Double` etc.

The following methods are responsible for **outgoing messages**:

- `sendMessageToBus(String message, String recipient)`: A message to the imaginary communications "bus" that the experiment participant is connected

to. First it goes to the control center for evaluation and logging, and from there it is distributed to the final recipient (which can also be "all", in order to reach all present participants).

- `sendStatusMessage(String message)`: Status messages can contain data for experiment evaluation. They are simply sent to the control center and written to the log file.

And in the opposite direction, **incoming messages**:

- `receivedMessageOnBus(String sender, String content)`: Callback method, delivering a message from another experiment participant, the `sender`.

5.3.1.4 Multi Threading Problems

In the way presented above, the experiment participant functionalities are only suitable for applications that do not perform heavy computational tasks. If that were the case, all tasks that are also running in the main thread, will be slowed down or even halted, because the main thread is also where the actual "simulation work" of the application is done. The most common occurrence of this problem is when the display "freezes" and thus behaves strangely: Buttons do not react, menus stay open instead of collapsing or other obsolete information is being displayed. The interaction with the "outside world" (i. e. all RTI related business) however will not be affected because all related tasks are already provided with their own threads.

These issues are not resolved easily and it has to be considered whether the CMIF HLA Environment is the right testbed for "number crunching" applications, anyway.

Nevertheless, there is one possibility how to decouple the simulation task from from the GUI related work and thus make the display perform smoothly without in-depth knowledge of Java threads programming. The utility class `SwingWorker` (see appendix D.4.3) provides an easy means for using an additional thread to keep the display from halting. Just place the following code whereever the new thread is needed:

```
final SwingWorker worker = new SwingWorker() {
   public Object construct() {

      //...code that might take a while
      //to execute is here...

      return null;
   }
};
worker.start();
```

The call to `worker.start()` initiates the new thread execution, but then immediately returns from the call and continues in the old thread. An extra thread can be very convenient, but the following limitations exist: From inside the new thread you cannot perform any GUI related work, except for calls to `Container.revalidate()`, `Component.repaint()` because only those are handled in a thread-safe way.

There are much more functionalities to the `SwingWorker` class, though. A detailed description can be found at [Jav00a], under `/uiswing/misc/threads.html`.

5.3.2 An Example: ContactGenerator and TacticDisplay

To do a "Proof-of-Concept" for the CMIF HLA Environment implementation, as well as to evaluate handling of the control center and test system performance, two demo

applications were implemented. First, the functionalities of the `ContactGenerator` and `TacticDisplay` will be described. Then it will be shown how the concepts, methods and callbacks which were described in the last section, are actually put to work in "real" applications. However, the `ContactGenerator` as well as the `TacticDisplay` do not contain any functionalities related to actual distributed data fusion.

5.3.2.1 The TacticDisplay Application

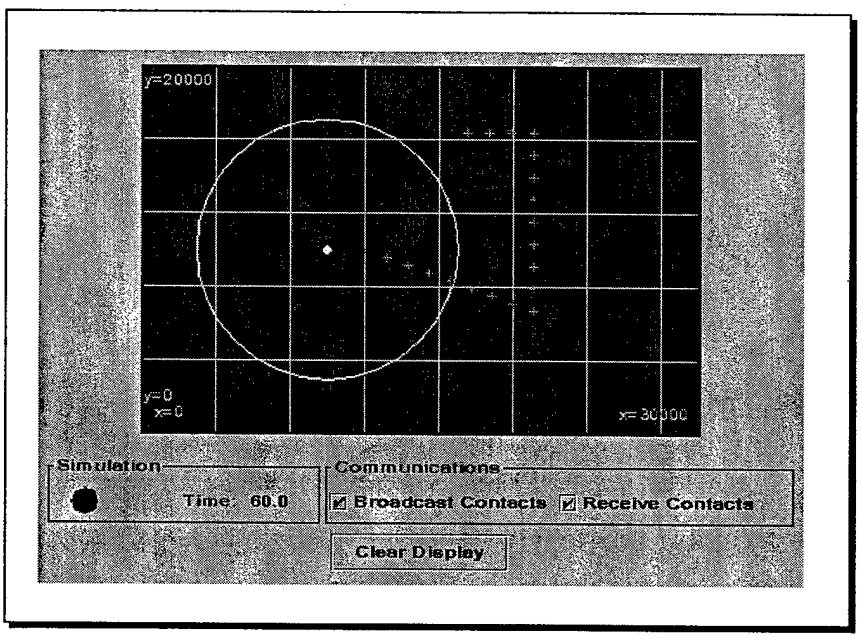

Figure 5.3: Screenshot of the `TacticDisplay` application user interface

The `TacticDisplay` resembles a simple radar display. Its user interface features a radar screen with 2-D coordinates. Within the rectangular display area, a yellow circle indicates the imaginary "radar range" of the `TacticDisplay`. During the simulation run, the application reacts to incoming messages, which resemble position reports. First-hand radar contacts are reported to the application in the following way:

`ContactX=#11705#Y=#2342#.`

The program now evaluates if an incoming report falls within the range of its own radar, if it can actually "see" the target itself. If that is the case, the position report is displayed on the screen with a red `x`. Furthermore, any `TacticDisplay` also shares known contacts by rebroadcasting them with a message like this:

`ReportedContactX=#11705#Y=#2342#.`

Any other `TacticDisplay`, which is present as an experiment participant, will display such a second-hand report with a grey `+`, no matter, if the position falls into its own reception range or not.

In addition, the application also has a "Communications" panel which lets the user toggle the behaviour regarding the broadcasting and the evaluation of second-hand reports. The "Simulation" panel just displays information about the state of the simulation run and the current federate time. A screenshot of the graphical user interface is shown in figure 5.3.

5.3.2.2 The ContactGenerator Application

As the name suggests, the `ContactGenerator` provides the `TacticDisplays` in the simulation space with the mentioned first-hand contact reports. To come up with these, it lets the user "fly" an imaginary "airplane" across the 2-D plane. The user interface again has a 2-D display area on which the current and former positions of the airplane are depicted. The flight path is determined by entering (and dynamically altering) values for the parameters `X Position`, `Y position`, `Velocity` and `Heading` in the according textboxes. Like in the `TacticDisplay`, the "Simulation" panel

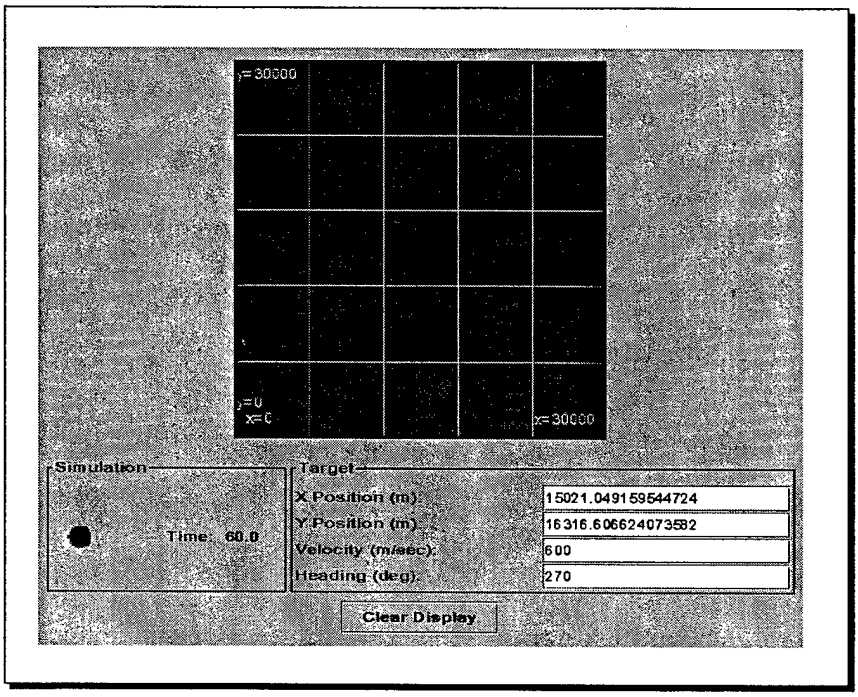

Figure 5.4: Screenshot of the `ContactGenerator` application user interface

informs about simulation state and progress of time. A depiction of the user interface can be seen in figure 5.4.

5.3.2.3 Implementation

The full sourcecode for both the `TacticDisplay` and the `ContactGenerator` class can be found in the appendix, sections D.3.3 and D.3.1, respectively. At this point however it will be shown, how the methods of the CMIF HLA Environment are to be used and implemented, in order to show the developer how to approach the implementation of own experiment participants.

First of all, the CMIF HLA Environment packages have to be made accessible using the import declarations at the beginning of the code:

```
import CMIF_HLA_Environment.CMIFExperimentParticipant;
import CMIF_HLA_Environment.util.*;
```

Then comes the class declaration, also defining, that the new class is a subclass to the abstract class `CMIFExperimentParticipant`:

```
public class ContactGenerator
  extends CMIFExperimentParticipant
```

The static method `main(String argv[])` is always called by the Java virtual machine at the start of any application. Command line arguments, if any, are passed in the `String` array `argv[]`. Because we want an object instance of our application, we create it right there, using the `new` operator:

```
public static void main(String argv[]) {
  ContactGenerator instance =
    new ContactGenerator(argv);
}//end of main
```

Now follows the constructor, which could be considered the start-up method of the object instance. The first method call has to go to the constructor of the superclass, using `super()`. In this case, `CMIFExperimentParticipant` is the superclass and it expects one `String` argument, the name of the specific experiment participant by which it later will be identified within the simulation space. For the `ContactGenerator` the name is fixed, whereas the `TacticDisplay` adapts the first command line argument as its name during startup.

```
public ContactGenerator(String[] argv) {
  super("ContactGenerator");
  ...
  ...
```

Now, still at the beginning of the code, the crucial steps to get involved with the rest of the simulation space follow like this: First the setup phase is initiated and then, while that is taken care of asynchronously in the background, the method `createDisplay()` is called. This method is application specific and contains the code to put together and initialize the graphical user interface. With Java and Swing, Java's GUI toolkit, this itself can take some time and might be rather slow. Normally, a callback from the superclass (`doneWithSetup()`) indicates when the setup phase has successfully been completed, but since we do not need to do anything special in that case, we just enter the method `waitForSimulation()`, which holds the program execution until the control center gives the signal that the simulation run has started.

```
    ...
    ...
    doSetup();
    createDisplay();
    waitForSimulation();
    startSimulation();
  }//end of constructor
```

Everything that has to be done during the run, has been moved to the method `startSimulation()`. In our case, `ContactGenerator` is the Time-Stepped simulation component, because once during every step it will calculate a new position for the imaginary airplane based on the old position and the current settings of the textboxes. The new position will then be displayed and sent out as a message. All this is done within a `while`-loop which executes for the duration of the simulation. At the end of each `while` iteration, the call to `advanceOneStep()` ensures, that the simulation time is being advanced. So the simulation method contains the following code:

```
public void startSimulation() {
```

```
    ...

    while (simIsRunning()) {

      determineNextPosition();
      displayPanel.repaint();
      sendContactPosition();
      advanceOneStep();

    }//end of while loop
  }//end of startSimulation
```

The TacticDisplay is by its nature, i. e. reacting to external input, an Event-Based experiment participant. Nevertheless, it too has the charateristic simulation while-loop, but the only task of this loop is to follow the simulation time and correctly display it for the user.

Now we will have a closer look at how the communications are taken care of. Whenever a message is delivered to a TacticDisplay, this is done by a callback to the following method:

```
public void receivedMessageOnBus(String sender,
                                  String content) {

  //first dissect the message content
  //and store the coordinates in a point

  //...if it is a first_hand contact: test if in radar range
  boolean isWithinRange = evaluateRadarRange(contactPoint);
  if (isWithinRange) {

      //add to the list of previous first-hand contacts
      //(used for drawing the x's)
      radarContacts.add(contactPoint);

      //and rebroadcast the same position report, only
```

```
        //with "Reported" added to the message header
        sendMessageToBus("Reported"+content, "all");
    }

    //...if it is a second-hand contact, add the point
    //to the list of previous second-hand contacts
    //(used for drawing the +'s)
    reportedContacts.add(contactPoint);

}//end of receivedMessageOnBus
```

The code snippet has been abbreviated to show only the important steps. Most interesting is firstly how to generally fill the `receivedMessageOnBus` method with life and secondly how to use the `sendMessageToBus` method, in this case with `"all"` as the desired recipients for the message. That means, also the `ContactGenerator` will get the message, only that it will not evaluate and use it, since it does not have an overriden version of `receivedMessageOnBus`.

In both applications, the other callback methods —`updateSimTime()`, `timingParametersChanged()`, `doneWithSimulation()`, etc.— do not do much except to force the user interface to show updated information on the "Simulation" panel.

Last not least, the callback indicating that the contact to the RTI services and therfore all other experiment participants has successfully been terminated, leads to the final shut down of the application.

```
public void doneWithShutDown(){

  mainWindow.dispose();
  System.exit(0);

}//end of  doneWithShutDown
```

5.4 Putting Everything to Work: A Simulation Run

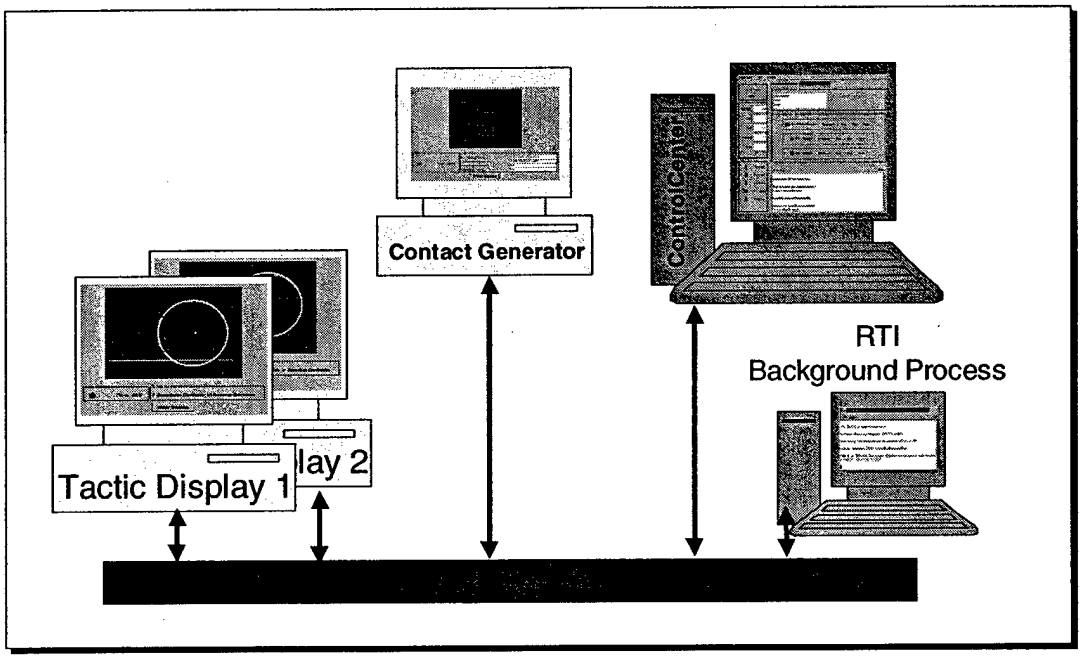

Figure 5.5: Structure of the demo application

In the previous section, the implementation of two experiment participants was discussed. Now it will be shown, how these applications are started and, together with the other components of the infrastraucture, are integrated into the simulation space to form an experiment setup. Then, we will go through the actual process of conducting a simulation run, step by step from startup to shutdown. Whenever specific commands are listed, these apply to using the CMIF HLA Environment on the UNIX (or Linux) platform. The general handling of the system is not different for the Windows operating systems, although minor adjustments have to be made.

As can be seen in figure 5.5, the configuration will feature one `ContactGenerator`

and two TacticDisplays. These will both have the same display area, but different areas of "radar reception" so that the benefit of their data sharing behaviour can be observed. Also present, of course, will be the RTI background services as well as the control center. All components will be interconnected through the local area network.

The first step for running an experiment within the CMIF HLA Environment is always to start the RTI background services. This is done by starting the rtiexec process on one of the computers that are involved. Just type

```
kharth@kadar> $RTI_HOME$RTI_BUILD_TYPE/bin/rtiexec
-endpoint <hostname>:<port number>
```

at a command line prompt. The parameters <hostname> and <port number> of the -endpoint option must hereby match those defined in the RTI.rid file or an error will occur (see section 5.1). It can also be useful to define a shortcut for this, because this line will be used quite often. As a result, the RTI services are initialized and remain in the background for all federates to use. The rtiexec is textbased. After startup it displays only messages about created and destroyed federation executions and joined or leaving federates. Also, system level error messages are shown, in case something goes wrong. See figure 5.6 for a sample output of rtiexec.

Now the control center has to be started, prior to all experiment participant components, to work on the experiment configuration and later, start the federation execution. First change to the root directory of the CMIF HLA Environment:

```
kharth@kadar> cd $JAVA_BINDING_HOME/apps
```

(or add that path to your Java environment variable CLASSPATH, for convenience) and then start Java:

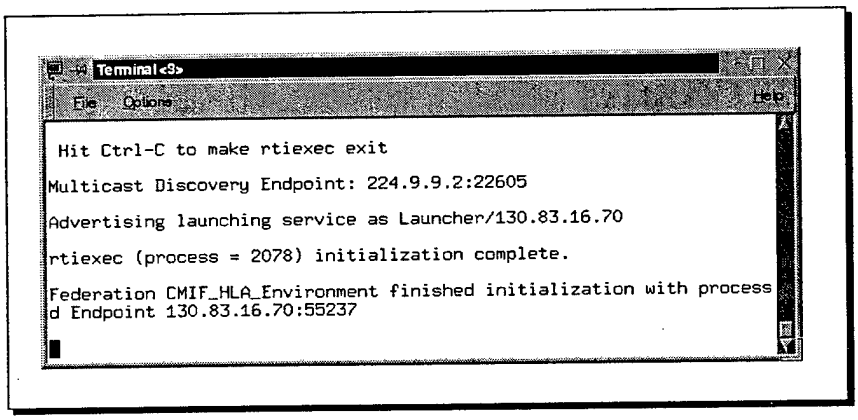

Figure 5.6: Before experiment participants and the control center can hook up with each other, the background process of the RTI, `rtiexec`, has to be present. It is launched from the command line of a terminal window.

```
kharth@kadar> java -native CMIF_HLA_Environment/
CMIFControlCenter/CMIFControlCenter
```

Phase 1: Configuration

Once the main window appears, the experiment configuration can be begun: Open a new experiment, using the menu `Experiment>New`, or alternatively, open an exitsting one with `Experiment>Open Old Experiment`. If you chose `New`, you will be prompted for an experiment name and a short description. While the latter is optional, the name will also be used as a file name for saving the configuration.

The most important step is to define the list of experiment participants to be expected in the simulation space. Because the control center is supposed to be independent of specific experiment setups, it can only be informed at runtime, which applications will be participating, in order to contact and address them properly when they arrive in the simulation space. We do this by simply adding the names of the desired experiment participants to the "Participants to Contact" list. This time it

will be `ContactGenerator`, `Display1` and `Display2`.

Now we will adjust the experiment timing parameters for our purposes by simply editing the textboxes in the "Time Management" panel: Let us choose `60.0` seconds as a new end time for the simulation and set the time step increment to `2.0` seconds. Important: Changes do not have any effect until the "Update" button is clicked.

With the configuration now completed, the settings can be saved using the menu item `Experiment>Save`. The experiment configuration file (`.cef`) belonging to this specific configuration can be found in the appendix, section C. Note that all the time the ongoing configuration phase has correctly been indicated by the blinking green icon in the "Phases" panel.

Phase 2: RTI Setup

By switching to the next phase (menu: `Phases>RTI Setup`), the user initiates the following actions:

- The control center creates the federation execution `fedex` process for the CMIF HLA Environment. It also joins the federation (and thus the "simulation space") as the first participant.
- Now the control center waits for other federates to become present, asks for their names and compares them to the list of experiment participants to be contacted.
- Once all the expected experiment participants are located, contacted and have reported ready for simulation, the control center declares the RTI setup phase for complete.

This is the point where all other experiment participants have to be launched. Generally, these are independent from each other and from the control center. They run in separate processes and can (and in fact should) be located on different computers. The only rule is, that any experiment participant's `doSetup()` must not be called before the control center initiated the federation execution, because the experiment participants just rely on the `fedex` to be present. Now we can type (in different terminal windows or on different machines):

```
kharth@kadar> java -native CMIF_HLA_Environment/
CMIFExperimentParticipant/ContactGenerator

kharth@ming> java -native CMIF_HLA_Environment/
CMIFExperimentParticipant/TacticDisplay Display1 0 0
30000 20000 10000 10000 8500

kharth@joker> java -native CMIF_HLA_Environment/
CMIFExperimentParticipant/TacticDisplay Display2 0 0
30000 20000 20000 10000 8500
```

Here is a key to the long list of command line parameters for the `TacticDisplay`:

- Name of this experiment participant. This must match the entry in the control center's list of participants to contact.
- X and Y coordinates of the lower left corner of the display area, in "real life" units
- Width and Height of the display area
- X and Y coordinates of the radar sensor
- Radius of the circular area of radar reception

Again, it might be useful to put these long commands into batch files, to avoid having to type them each time anew.

While waiting for the experiment participants to set up their contact to the RTI and their display components, we can watch how the control center, one by one, discovers the experiment participants' presence and lists them on the "During Simulation..." panel. The whole process might take a couple of seconds, depending on system performance.

After all the negotiations have been completed and all experiment participants reported that they are ready for the simulation run (indicated by the steady green icon on their "During Simulation..." panel entry), the control center now declares the RTI Setup phase for completed, accordingly switches the icons on the "Phases" panel and allows a switching to the next phase.

Before starting the simulation, a more convenient setting for the virtual airplane is entered, starting position: `X = 0`, `Y = 10000`, `Velocity = 500` and `Heading = 90` to place it in the middle of the left display boundary, going straight to the right.

Phase 3: Simulation Run

The actual simulation execution is started with `Phases>Run Simulation`. The control center sends out the "Go" signal to all participating applications and starts providing the time steps which serve as master time for all experiment participants. In this case, there is a two second increase in simulation time for every two seconds in real time. Immediately, the experiment participants start counting their own time, which should be fairly synchronous to the master time. The `ContactGenerator` starts displaying and sending the current positions of the airplane. The two `TacticDisplays`

also start their displaying of first and second-hand contacts, usually lagging one time step behind the generation of the respective contacts.

During the 60.0 seconds of the simulation run, we can play around by altering the airplane's course and the `TacticDisplays`' behaviour regarding the treatment of "Reported Contacts".

After the Simulation

After the simulation time has elapsed, all participants stop their simulation actions and wait for further instructions. A red icon with the experiment participants indicates their current stopped status. Now the experiment participant applications can be killed with their remote shutdown buttons from the control center and the RTI shutdown phase can be initiated using the menu `Phases>RTI Shutdown`. After all RTI services have successfully been terminated, we can, as the last step, exit the control center.

5.5 A Few Practical Remarks about the Usage of the CMIF HLA Environment

In the previous section, a sample experiment execution has been discussed. This was to show how the CMIF HLA Environment and the control center are supposed to (and in the described configuration actually do) work together. However, with this very first version of the software, the author cannot guarantee, that it is free of errors, "dead ends" and illogical behaviour. It is likely, that you will encounter bugs or that components will "get stuck" in certain situations. While the work on the

CMIF HLA Environment will be continued –to implement some missing features as well as to eliminate as many of those present problems– right now you should follow the undermentioned rules in case something goes wrong:

- Since otherwise stable behaviour cannot be guaranteed, all participating porgrams have to be closed, also the control center and the background processes.
- Check if any processes got stuck, under UNIX with `ps -u <username>`, under Windows with the Task-Manager. Look for `rtiexec`, `fedex` and `java`. If neccessary, they have to be killed manually, using the `kill -9 <PID>` command. `<PID>`, in that case, is the process ID that was listed by the `ps` command.
- In case of a violent crash, Java writes `core` files. These are quite large and should simply be deleted.

If other errors or bugs become known during the course of the development of a new application that utilizes the CMIF HLA Environment, please contact the author at `Kai.Harth@gmx.de`.

Chapter 6

Conclusion

6.1 Assessment of Results

During the implementation of the CMIF HLA Environment, it became more and more obvious, that the High Level Architecture is a highly sophisticated and professional product. It proved to be well thought out on the conceptual level (HLA) as well as on the technical level of the RTI implementation. With its enormous versatility and flexibility, the problem for a small-scale project like this was rather to grasp what the RTI all can do, than to actually make it work according to one's own intentions. In other words, retardations were due to initial misconceptions of the functioning and not to weaknesses in the HLA concept. In this regard it is helpful, that the RTI is really well documented.

Although the author makes no claim whatsoever to fully understand and utilize all concepts and capabilities of the HLA/RTI, it seems like the implementation of

the CMIF HLA Environment was still reasonably successful. The first version seems to live up to its purposes and provide most of the functionalities that were initially planned. The control center offers a graphically appealing and mostly self-explanatory means for the user to operate a distributed simulation experiment within the CMIF HLA Environment. And the application programming toolkit provides the developer with access to the RTI resources while maintaining the intended simplicity.

The following table gives a final overview over the mapping between HLA and the CMIF HLA Environment. It is summarized, how the various concepts of the High Level Architecture are implemented and substantiated.

High Level Architecture	**CMIF HLA Environment**
Objects	
any (defined by FOM)	$(1 \ldots n)$ `CMIFExperimentParticipant`, 1 `RTIControlModule`
Interactions	
any (defined by FOM)	`MessageToBus`, `MessageFromBus`, `ControlInteraction`, `StatusMessage`
Timing	
Various timing concepts can be implemented (ref. figure 2.2).	Time-scale synchronization with variable scaling factor Sc and variable step increment.
...	...

(High Level Architecture)	(CMIF HLA Environment)
...	...
Timing control is handled flexibly by having federates declare to be `time-Regulating` and `timeConstrained`.	"Master-Slave" concept: The control center is `timeRegulating` and the experiment participants are `timeConstrained`.
Communications	
Communications are implemented by having federates issue interactions, all common basic data types are usable.	Experiment participants can communicate by sending `MessageToBus` and receiving `MessageFromBus`, while the control center exerts remote control by issuing `ControlInteractions`. (data type is always `String`)
Simulation Behaviour	
Event-Based and/or Time-Stepped	Event-Based and/or Time-Stepped
Support Tools	
Various support tools are available [Def00a].	The "CMIF Control Center", featuring functionalities for experiment configuration, control and (in a later version) evaluation. The control center is only usable within the CMIF HLA Environment.
...	...

(High Level Architecture)	(CMIF HLA Environment)
Platforms	
Numerous system platforms and several programming languages.	All platforms, for which a RTI Java binding exists. The programming language is limited to Java.
Summary	
Flexibility and versatility are high.	Flexibility and versatility are reduced, but tailored to CMIF needs.
Complexity is high.	Complexity is extremely reduced.

6.2 Future Work

Although a proof-of-concept has been done for the CMIF HLA Environment, this was only the first step towards a validation of the software. The `DemoApplication` did show, that the CMIF API is in fact usable and that the control center does work –but only on a very undemanding level. Now the software should prove its workableness through the implementation of the first "real" distributed experiment. One that contains actual functionality and is focused on coming up with real results, and not on testing the CMIF HLA Environment functionalities.

Also some work needs to be done on the control center application. Besides the removal of several internal errors, there are still some intregral functionalities, that were planned in the initial concept, but could not be implemented yet. Among these are:

- Generation of a log file (communications and additional information) during the simulation run.
- Evaluation (and visualization) of the amount of data being transferred between the experiment participants.
- Advanced control over the inter experiment participant messaging, e. g. emulation of "bandwidth" limitations; advanced routing possibilities.

Only after these steps have been taken will the CMIF HLA Environment be a fully developed and usable software package. Then, hopefully, the CMIF HLA Environment will become the basis for and a part of numerous distributed simulation experiments at CMIF in the future.

Appendix A

The RTI Configuration File RTI.rid

```
;; This is an example of a small RID file.  Please see Sample-RTI.rid
;; from the distribution for an exhaustive RID file which includes
;; descriptions of the parameters.

(RTI
 (ProcessSection
  (RtiExecutive
   (RtiExecutiveEndpoint kadar.eng.buffalo.edu:12345)

   ;; remember that rtiexec -endpoint flag must
   ;; match this, or you'll get NameService errors

  )
 )

 (FederationSection
  (Networking
   (BundlingOptions
    (UDP
     ;;(MaxTimeBeforeSendInSeconds 0.005)
     ;;(MaxBytesBeforeSend    63000)
    )
    (TCP
     ;;(MaxTimeBeforeSendInSeconds 0.005)
     ;;(MaxBytesBeforeSend    63000)
    )
   )
   (MulticastOptions
    ;; having different federations on network use different ranges of
    ;; multicast addresses will help performance
    (BaseAddress 224.100.0.0)
    ;;(MaxAddress 239.255.255.255)
   )
  )

  (Advisories
   ;;(RelevanceAdvisoryAttributeInstanceHeartbeatInSeconds Off)
```

```
    ;;(RelevanceAdvisoryAttributeInstanceTimeoutInSeconds Off)
    ;;(RelevanceAdvisoryInteractionClassHeartbeatInSeconds Off)
    ;;(RelevanceAdvisoryInteractionClassTimeoutInSeconds Off)
    ;;(RelevanceAdvisoryObjectClassHeartbeatInSeconds Off)
    ;;(RelevanceAdvisoryObjectClassTimeoutInSeconds Off)
  )

  (FederateSection

        ;;PARAMETER: FederateSection.AllowReentrantUpdatesAndInteractions
        ;;DESCRIPTION: If this option is enabled, federates will be able
        ;;to invoke updateAttributesValues, sendInteraction,
        ;;requestClassAttributeValueUpdate, and
        ;;requestObjectAttributeValueUpdate from within FederateAmbassador
        ;;callbacks.
        ;;RANGE: An enumeration value {Yes, No}.
        ;;DEFAULT_VALUE: No
        (AllowReentrantUpdatesAndInteractions Yes)

  (EventRetractionHandleCacheOptions
        ;; the next two options will disable event retractions, which is
        ;; OK since helloworld doesn't use them
        (MinimumCacheSizeBeforePerformingPurge 0)
        (NumberOfEventRetractionHandlesToCreateBeforeStartingNewPurgeCycle 0)
        )
    )
  )
)
```

Appendix B

The Federation Configuration File CMIF_HLA_Environment.fed

```
(FED
  (Federation CMIF_HLA_Environment)
  (FEDversion v1.3)
  (spaces
    (space thefinalfrontier
      (dimension x)
      (dimension y)
      (dimension z)
      (dimension time)
    )
    (space a
       (dimension x)
       (dimension y)
    )
    (space b
        (dimension x)
        (dimension y)
        (dimension z)
    )
    (space c
        (dimension x)
    )
    (space ip_space
       (dimension subnet)
    )
  )
  (objects
    (class ObjectRoot
      (attribute privilegeToDelete reliable timestamp A)
      (class RTIprivate)
      (class Manager
        (class Federate
          (attribute FederateHandle reliable receive)
          (attribute FederateType reliable receive)
          (attribute FederateHost reliable receive)
          (attribute RTIversion reliable receive)
          (attribute FEDid reliable receive)
```

```
            (attribute TimeConstrained reliable receive)
            (attribute TimeRegulating reliable receive)
            (attribute AsynchronousDelivery reliable receive)
            (attribute FederateState reliable receive)
            (attribute TimeManagerState reliable receive)
            (attribute FederateTime reliable receive)
            (attribute Lookahead reliable receive)
            (attribute LBTS reliable receive)
            (attribute MinNextEventTime reliable receive)
            (attribute ROlength reliable receive)
            (attribute TSOlength reliable receive)
            (attribute ReflectionsReceived reliable receive)
            (attribute UpdatesSent reliable receive)
            (attribute InteractionsReceived reliable receive)
            (attribute InteractionsSent reliable receive)
          (class Federation
            (attribute FederationName reliable receive)
            (attribute FederatesInFederation reliable receive)
            (attribute RTIversion reliable receive)
            (attribute FEDid reliable receive)
            (attribute LastSaveName reliable receive)
            (attribute LastSaveTime reliable receive)
            (attribute NextSaveName reliable receive)
            (attribute NextSaveTime reliable receive) )
        )


;; user object classes here
        (class RTIControlModule
            (attribute ExperimentState reliable timestamp)
        )
        (class CMIFExperimentParticipant
            (attribute ParticipantName reliable timestamp)
            (attribute ParticipantState reliable timestamp)
        )
      )
    )
    (interactions
      (class InteractionRoot reliable receive
        (class RTIprivate reliable receive)
        (class Manager reliable receive
          (class Federate reliable receive
            (parameter Federate)
            (class Adjust reliable receive
              (class SetTiming reliable receive
                (parameter ReportPeriod) )
              (class ModifyAttributeState reliable receive
                (parameter ObjectInstance)
                (parameter Attribute)
                (parameter AttributeState) )
              (class SetServiceReporting reliable receive
                (parameter ReportingState) )
              (class SetExceptionLogging reliable receive
                (parameter LoggingState) )
            )
            (class Request reliable receive
```

```
    (class RequestPublications reliable receive)
    (class RequestSubscriptions reliable receive)
    (class RequestObjectsOwned reliable receive)
    (class RequestObjectsUpdated reliable receive)
    (class RequestObjectsReflected reliable receive)
    (class RequestUpdatesSent reliable receive)
    (class RequestInteractionsSent reliable receive)
    (class RequestReflectionsReceived reliable receive)
    (class RequestInteractionsReceived reliable receive)
    (class RequestObjectInformation reliable receive
      (parameter ObjectInstance) )
)
(class Report reliable receive
    (class ReportObjectPublication reliable receive
      (parameter NumberOfClasses)
      (parameter ObjectClass)
      (parameter AttributeList) )
    (class ReportInteractionPublication reliable receive
      (parameter InteractionClassList) )
    (class ReportObjectSubscription reliable receive
      (parameter NumberOfClasses)
      (parameter ObjectClass)
      (parameter AttributeList)
      (parameter Active) )
    (class ReportInteractionSubscription reliable receive
      (parameter InteractionClassList) )
    (class ReportObjectsOwned reliable receive
      (parameter ObjectCounts) )
    (class ReportObjectsUpdated reliable receive
      (parameter ObjectCounts) )
    (class ReportObjectsReflected reliable receive
      (parameter ObjectCounts) )
    (class ReportUpdatesSent reliable receive
      (parameter TransportationType)
      (parameter UpdateCounts) )
    (class ReportReflectionsReceived reliable receive
      (parameter TransportationType)
      (parameter ReflectCounts) )
    (class ReportInteractionsSent reliable receive
      (parameter TransportationType)
      (parameter InteractionCounts) )
    (class ReportInteractionsReceived reliable receive
      (parameter TransportationType)
      (parameter InteractionCounts) )
    (class ReportObjectInformation reliable receive
      (parameter ObjectInstance)
      (parameter OwnedAttributeList)
      (parameter RegisteredClass)
      (parameter KnownClass) )
    (class Alert reliable receive
      (parameter AlertSeverity)
      (parameter AlertDescription)
      (parameter AlertID) )
    (class ReportServiceInvocation reliable receive
      (parameter Service)
      (parameter Initiator)
```

```
      (parameter SuccessIndicator)
      (parameter SuppliedArgument1)
      (parameter SuppliedArgument2)
      (parameter SuppliedArgument3)
      (parameter SuppliedArgument4)
      (parameter SuppliedArgument5)
      (parameter ReturnedArgument)
      (parameter ExceptionDescription)
      (parameter ExceptionID) )
)
(class Service reliable receive
  (class ResignFederationExecution reliable receive
    (parameter ResignAction) )
  (class SynchronizationPointAchieved reliable receive
    (parameter Label) )
  (class FederateSaveBegun reliable receive)
  (class FederateSaveComplete reliable receive
    (parameter SuccessIndicator) )
  (class FederateRestoreComplete reliable receive
    (parameter SuccessIndicator) )
  (class PublishObjectClass reliable receive
    (parameter ObjectClass)
    (parameter AttributeList) )
  (class UnpublishObjectClass reliable receive
    (parameter ObjectClass) )
  (class PublishInteractionClass reliable receive
    (parameter InteractionClass) )
  (class UnpublishInteractionClass reliable receive
    (parameter InteractionClass) )
  (class SubscribeObjectClassAttributes reliable receive
    (parameter ObjectClass)
    (parameter AttributeList)
    (parameter Active) )
  (class UnsubscribeObjectClass reliable receive
    (parameter ObjectClass) )
  (class SubscribeInteractionClass reliable receive
    (parameter InteractionClass)
    (parameter Active) )
  (class UnsubscribeInteractionClass reliable receive
    (parameter InteractionClass) )
  (class DeleteObjectInstance reliable receive
    (parameter ObjectInstance)
    (parameter FederationTime)
    (parameter Tag) )
  (class LocalDeleteObjectInstance reliable receive
    (parameter ObjectInstance) )
  (class ChangeAttributeTransportationType reliable receive
    (parameter ObjectInstance)
    (parameter AttributeList)
    (parameter TransportationType) )
  (class ChangeAttributeOrderType reliable receive
    (parameter ObjectInstance)
    (parameter AttributeList)
    (parameter OrderingType) )
  (class ChangeInteractionTransportationType reliable receive
    (parameter InteractionClass)
```

```
            (parameter TransportationType) )
          (class ChangeInteractionOrderType reliable receive
            (parameter InteractionClass)
            (parameter OrderingType) )
          (class UnconditionalAttributeOwnershipDivestiture receive
            (parameter ObjectInstance)
            (parameter AttributeList) )
          (class EnableTimeRegulation reliable receive
            (parameter FederationTime)
            (parameter Lookahead) )
          (class DisableTimeRegulation reliable receive)
          (class EnableTimeConstrained reliable receive)
          (class DisableTimeConstrained reliable receive)
          (class EnableAsynchronousDelivery reliable receive)
          (class DisableAsynchronousDelivery reliable receive)
          (class ModifyLookahead reliable receive
            (parameter Lookahead) )
          (class TimeAdvanceRequest reliable receive
            (parameter FederationTime) )
          (class TimeAdvanceRequestAvailable reliable receive
            (parameter FederationTime) )
          (class NextEventRequest reliable receive
            (parameter FederationTime) )
          (class NextEventRequestAvailable reliable receive
            (parameter FederationTime) )
          (class FlushQueueRequest reliable receive
            (parameter FederationTime) )
        )
      )
    )


;;  user interaction classes here
      (class ControlInteraction reliable timestamp
        (parameter ControlMessageContent)
        (parameter ControlMessageRecipient)
      )
      (class MessageToBus reliable timestamp
        (parameter ToBusMessageContent)
        (parameter ToBusSender)
        (parameter ToBusRecipient)
      )
      (class MessageFromBus reliable timestamp
        (parameter FromBusMessageContent)
        (parameter FromBusSender)
        (parameter FromBusRecipient)
      )
      (class StatusMessage reliable timestamp
        (parameter StatusMessageContent)
        (parameter StatusMessageSender)
      )
    )
  )
)
```

Appendix C

A Sample Experiment Configuration (.cef) File

```
#=======================#
# CMIF_HLA_ENVIROMENT   #
# Experiment Data File  #
#=======================#

#Generated File! Edit only if you know what you are doing....

#general info

>>author{harth}

>>name{CMIFDemoApplication01}

>>description{A simple Demo Application involving
a "ContactGenerator", sending contact signals and
two "TacticDisplay"s, which receive the contact
information, display it and interact with each other}

# time management

>>start_time{0.0}
>>end_time{60.0}
>>interval_time{2.0}
>>scale{1.0}

# participants
```

```
>>participant_num{3}
>>participant{1, ContactGenerator}
>>participant{2, Display1}
>>participant{3, Display2}

#end of configuration file
```

Appendix D

Sourcecodes of the Java classes

D.1 The package CMIFControlCenter

D.1.1 The Class CMIFControlCenter.java

```
//-----------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                   /
//                                                     /
//This is part of a Master's Thesis at the             /
//Center for Multisource Information Fusion            /
//SUNY at Buffalo                                      /
//                                                     /
//All rights reserved:                                 /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth      /
//packages hla.rti13.java1      : U.S. DoD             /
//packages java, javax          : Sun Microsystems     /
//-----------------------------------------------------/

package CMIF_HLA_Environment.CMIFControlCenter;

import CMIF_HLA_Environment.util.*;
import CMIF_HLA_Environment.CMIFExperimentParticipant.*;

import hla.rti13.java1.*;

import java.awt.event.*;
import javax.swing.Box;


//=========================================================
//=========================================================

public class CMIFControlCenter

  implements ActionListener


{

  //The class CMIFControlCenter is the central class for the
  //CMIF HLa environment. It implements the "White" Control
```

```
//Center used to setup, control and monitor experiments.

//static variables:

public static int IDLE = 0;
public static int SETUP_EXPERIMENT = 1;
public static int SETUP_RTI = 2;
public static int RUN_EXPERIMENT = 3;
public static int CLEANUP_RTI = 4;
public static int POST_EXPERIMENT = 5;
public static int READY_FOR_SHUTDOWN = 6;

//instance variables:

public DebugHelper D = new DebugHelper ();

public CMIFControlDisplay CCDisplay;
public CMIFExperimentManager CCManager;
public RTIControlModule rModule;

public int cCenterState;


//=====================================================
//=====================================================


public static void main(String argv[]) {

  //The main() method is mandatory for starting a java
  //application. From here an instance of CMIFControlCenter
  //will be instanciated.


  CMIFControlCenter CCenter = new CMIFControlCenter();

  CCenter.init();


}//end of main

//=====================================================
//=====================================================

public CMIFControlCenter () {

  //Empty contructor to create an instance of CMIFControlCenter

  D.dbgOut("Started CMIFControlCenter.....");
  D.setDebugState(DebugHelper.FULL_DEBUG);

}//end of contructor

//=====================================================
//=====================================================

public void init() {

  //init() will organize:
  //-instanciation of all major parts of the CMIFControlCenter
  // Experiment manager, display

  D.dbgOut("$#Method: CMIFControlCenter.init()");
```

```
    CCManager = new CMIFExperimentManager(this);

    CCDisplay = new CMIFControlDisplay(this);

    rModule = new RTIControlModule(this);

    cCenterState=CMIFControlCenter.IDLE;

    CCDisplay.messageOut("Welcome to the CMIF HLA Control Center!");
    CCDisplay.messageOut("");
    CCDisplay.messageOut("To get started either load an existing experiment");
    CCDisplay.messageOut("or create a new experiment setup...");


  }//end of init()

  //=======================================================
  //=======================================================

  private boolean readyForShutdown() {

    //This method evaluates cCenterState and returns
    //whether the Control Center is in a state that allows
    //shutdown...

    boolean allow = false;

    allow = (cCenterState==IDLE);

    //for the time being...
    allow = true;

    return  allow;

  }//end of readyForShutdown

  //=======================================================
  //=======================================================

  private void exitControlCenter() {

    //This Method coordinates the final shutdown of the
    //control center. It has to be checked if all conditions
    //are met to allow the clean shutdown

    if (readyForShutdown()) {

      D.dbgOut("$Method: CMIFControlCenter.exitControlCenter()");

      rModule.runTickThread=false;

      CCDisplay.dispose();

      System.exit(0);

    } else {

      CCDisplay.messageOut("The current state of the control center does not");
      CCDisplay.messageOut("allow shutdown!");
      CCDisplay.messageOut("Please abort and close the current "+
                           "experiment first.");
```

```
  }
}//end of exitControlCenter
//-------------------------------------------
//-------------------------------------------
//
// Listener
//
//-------------------------------------------
public void actionPerformed(ActionEvent event) {
  String command = event.getActionCommand();
  D.dbgOut("ControlCenter.actionPerformed() received: " + command);
  if (command.equals("exit_control_center")){
    exitControlCenter();
  }
  //------------------------------
  if (command.equals("clear_messages")){
    CCDisplay.messageOut("%flush");
  }
  //------------------------------
  //------------------------------
  if (command.equals("display_about_text")){
                          CCManager.displayAboutText();
  }
  //------------------------------
  if (command.equals("new_experiment")){
    CCManager.configureNewExperiment();
  }
  //------------------------------
  if (command.equals("save_experiment")){
    ExperimentFileHandler.saveExperimentFile(CCManager);
  }
  //------------------------------
  if (command.equals("open_experiment")){
    ExperimentFileHandler.loadExperimentFile(CCManager);
  }
  //------------------------------
  //------------------------------
  //phases:
  if (command.equals("configure_experiment")){
```

```
  if (cCenterState==POST_EXPERIMENT) {

    cCenterState=SETUP_EXPERIMENT;
    CCDisplay.setPhaseIcons("config", "grn_pulse");
    CCDisplay.setPhaseIcons("simulation", "red");
    CCDisplay.setPhaseIcons("rticleanup", "red");

  } else if (cCenterState==IDLE || cCenterState==READY_FOR_SHUTDOWN) {
    CCDisplay.messageOut("");
    CCDisplay.messageOut("Please open a new or existing experiment!");
  } else if (cCenterState==SETUP_EXPERIMENT) {
    CCDisplay.messageOut("");
    CCDisplay.messageOut("Already there...");
  } else  {
    CCDisplay.messageOut("");
    CCDisplay.messageOut("The current state does not allow this "+
                         "operation!");
  }

  CCDisplay.switchToState( cCenterState );
}

//------------------------------

if (command.equals("run_rti_setup")){

  if (cCenterState==POST_EXPERIMENT) {

    cCenterState=SETUP_RTI;
    CCDisplay.setPhaseIcons("config", "grn");
    CCDisplay.setPhaseIcons("rtisetuo", "grn_pulse");
    CCDisplay.setPhaseIcons("simulation", "red");
    CCDisplay.setPhaseIcons("rticleanup", "red");

  } else if (cCenterState==SETUP_EXPERIMENT) {

    if (CCManager.configComplete()) {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("Attempting to set up the RTI and "+
                           "all participants....");
      CCDisplay.messageOut("This could take a few moments.");
      CCDisplay.setPhaseIcons("rtisetup", "grn_pulse");
      CCDisplay.setPhaseIcons("config", "grn");

      cCenterState=SETUP_RTI;

      //CCDisplay.phasesInnerPanel.validate();

      CCDisplay.pInnerPanel.removeAll();
      CCDisplay.pInnerPanel.add(Box.createVerticalGlue());

      final SwingWorker worker = new SwingWorker() {
        public Object construct() {

          //...code that might take a while
          //to execute is here...

          rModule.launchRTI();
          rModule.runPreSimulation();
```

```
            if (rModule.rtiState==rModule.RTI_RUNNING) {

              CCDisplay.setPhaseIcons("rtisetup",
                                      "grn");
              CCDisplay.setPhaseIcons("simulation",
                                      "orange");
              CCDisplay.setPhaseIcons("rticleanup",
                                      "orange");

            }
            return null;
          }
        };
        worker.start();

      } else {

        CCDisplay.messageOut("");
        CCDisplay.messageOut("The experiment configuration is not "+
                             "complete yet.");
        CCDisplay.messageOut("Please make sure you defined:");
        CCDisplay.messageOut("--> time management,");
        CCDisplay.messageOut("--> participants and");
        CCDisplay.messageOut("--> communications.");

      }

    } else if (cCenterState==IDLE || cCenterState==READY_FOR_SHUTDOWN) {
      CCDisplay.messageOut("");
      CCDisplay.messageOut("Please open a new or existing experiment!");

    } else if (cCenterState==SETUP_RTI) {
      CCDisplay.messageOut("");
      CCDisplay.messageOut("The RTI setup is already in progress.."+
                           "this might take a while!");

    } else  {
      CCDisplay.messageOut("");
      CCDisplay.messageOut("The current state does not allow "+
                           "this operation!");
    }

    CCDisplay.switchToState( cCenterState );

  }

  //------------------------------

  if (command.equals("run_simulation")){

    if ((cCenterState==SETUP_RTI) && (rModule.simState==rModule.SIM_READY) ) {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("Trying to start the simulation...");

      CCDisplay.setPhaseIcons("rtisetup", "grn");
      CCDisplay.setPhaseIcons("simulation", "grn_pulse");

      cCenterState=RUN_EXPERIMENT;
```

```
      //...what else??

      rModule.runSimulation();


    } else {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("The current state does not allow "+
                           "this operation!");

    }
  }

  //------------------------------
  //------------------------------

  if (command.equals("abort_simulation")){

    if (cCenterState==RUN_EXPERIMENT) {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("Aborting the simulation...");

      CCDisplay.setPhaseIcons("rtisetup", "grn");
      CCDisplay.setPhaseIcons("simulation", "orange");
      CCDisplay.setPhaseIcons("rticleanup", "orange");

      cCenterState=POST_EXPERIMENT;

      //...what else??

      rModule.abortSimulation();


    } else {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("The current state does not allow "+
                           "this operation!");

    }
  }
  //------------------------------

  if (command.equals("cleanup_rti")){

    if (cCenterState!=POST_EXPERIMENT) {

      CCDisplay.messageOut("");
      CCDisplay.messageOut("Shutting down the RTI...");

      CCDisplay.setPhaseIcons("rtisetup", "orange");
      CCDisplay.setPhaseIcons("simulation", "red");
      CCDisplay.setPhaseIcons("rticleanup", "grn_pulse");


      rModule.rtiState=rModule.RTI_SHUTDOWN;

      while(rModule.rtiState!=rModule.RTI_DOWN) {
      }
```

```
        cCenterState=IDLE;

      } else {

        CCDisplay.messageOut("");
        CCDisplay.messageOut("The current state does not allow "+
                             "this operation!");

      }
    }


  }//end of actionPerformed

};//end of class CMIFControlCenter
```

D.1.2 The Class CMIFControlDisplay.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth        /
//packages hla.rti13.java1      : U.S. DoD               /
//packages java, javax          : Sun Microsystems       /
//-------------------------------------------------------/

package CMIF_HLA_Environment.CMIFControlCenter;

import CMIF_HLA_Environment.util.*;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

import java.util.Properties;


//============================================================
//============================================================


public class CMIFControlDisplay extends JFrame

{

  //The class CMIFControlDisplay provides the ControlCenter with all
  //necessary display elements, sets up the main frame and keeps track of
  //updated etc.

  //instance variables

  CMIFControlCenter parentCCenter;

  DebugHelper D = new DebugHelper();

  public JMenuBar menuBar;

  JMenu experimentMenu, editMenu;
  public JMenu phasesMenu;

  JSplitPane mainSplitPane, rightSplitPane, leftSplitPane, participantsSplitPane;

  JTabbedPane topRightTabbedPane;

  JPanel  channelsPanel, communicationsPanel,
    timePanel, phasesPanel, messagePanel, upperLeftPanel, infoPanel,
    pConfigPanel, pSimulationPanel, pInnerPanel;

  public JPanel phasesInnerPanel;
```

```
public JLabel infoLabel;

JTextArea messageTextArea;

public JTextField startTimeTF, endTimeTF, intervalTF,
  scaleTF, masterTimeTF;


Dimension screenDimension, windowDimension;
             Point windowPosition;

//icons and bullets

String path = new String("CMIF_HLA_Environment/CMIFControlCenter/icons/");

ImageIcon greenIcon = new ImageIcon(path+"grn.gif");
ImageIcon greenIconPulse = new ImageIcon(path+"grn_pulse.gif");
ImageIcon redIcon = new ImageIcon(path+"red.gif");
ImageIcon redIconPulse = new ImageIcon(path+"red_pulse.gif");
ImageIcon orangeIcon = new ImageIcon(path+"orange.gif");
ImageIcon orangeIconPulse = new ImageIcon(path+"orange_pulse.gif");

Border lineBorder = BorderFactory.createLineBorder(Color.black, 2);
Border emptyBorder = BorderFactory.createEmptyBorder(8,8,8,8);

JLabel configLabel, rtiSetupLabel, simulationLabel, rtiCleanupLabel,
  shutdownLabel;

JScrollPane listScrollPane;

JButton updateButton, addButton, modifyButton,  deleteButton;

//=====================================================
//=====================================================

public CMIFControlDisplay (CMIFControlCenter parent) {

  //Constructor to create an instance of CMIFControlDisplay.
  //The superclass JFrame is instanciated, the control center
  //is made accessible by keeping a handle (parentCenter) to it
  //and the method createMainWindowLayout() is invoked.

  super("CMIF HLA Control Center");

  parentCCenter = parent;

  D.dbgOut("Started CMIFControlDisplay.....");
  D.setDebugState(DebugHelper.NO_DEBUG);

  createMainWindowLayout();

}//end of contructor

//=====================================================
//=====================================================

private void createMainWindowLayout() {

  //This mehtod creates the main window and its top level components,
  //the splitpanes etc. and then delegates to the methods responsible
  //for all the subcomponents
```

```
    D.dbgOut("$Method CMIFControlDisplay.createMainWindowLayout()");

    getContentPane().setLayout(new BorderLayout());

    createMenuBar();

    mainSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
    mainSplitPane.setDividerSize(10);

    rightSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    rightSplitPane.setDividerSize(5);

    leftSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    leftSplitPane.setDividerSize(0);
    leftSplitPane.setBorder(emptyBorder);

    getContentPane().add(mainSplitPane, BorderLayout.CENTER);

    createLeftSplitPane();

    createRightSplitPane();

    mainSplitPane.add(leftSplitPane);
    mainSplitPane.add(rightSplitPane);


    setComponentsSizes();


    //pack();
    show();

  }//end of createMainWindowLayout()

  //=========================================================
  //=========================================================

  private void createMenuBar() {

    //This Method creates the components and puts together
    //the control center's menu bar.
    //It has the following entries:
    //-Experiment
    //   >New
    //   >Open
    //   >Close
    //   >Save
    //   >Exit
    //-Edit
    //   >

    D.dbgOut("$Method CMIFControlDisplay.createMenuBar()");

    menuBar = new JMenuBar();

    //-----------------
    //Experiment menu:

    experimentMenu = new JMenu("Experiment");

    JMenuItem newMenuItem = new JMenuItem("New");
    JMenuItem openMenuItem = new JMenuItem("Open Old Experiment");
```

```
JMenuItem closeMenuItem = new JMenuItem("Close Current Experiment");
JMenuItem saveMenuItem = new JMenuItem("Save Configuration");
JMenuItem saveAsMenuItem = new JMenuItem("Save As...");
JMenuItem exitMenuItem = new JMenuItem("Exit");

newMenuItem.setActionCommand("new_experiment");
newMenuItem.addActionListener((ActionListener)parentCCenter);
openMenuItem.setActionCommand("open_experiment");
openMenuItem.addActionListener((ActionListener)parentCCenter);
closeMenuItem.setActionCommand("close_experiment");
closeMenuItem.addActionListener((ActionListener)parentCCenter);
saveMenuItem.setActionCommand("save_experiment");
saveMenuItem.addActionListener((ActionListener)parentCCenter);
saveAsMenuItem.setActionCommand("save_experiment_as");
saveAsMenuItem.addActionListener((ActionListener)parentCCenter);
exitMenuItem.setActionCommand("exit_control_center");
exitMenuItem.addActionListener((ActionListener)parentCCenter);


experimentMenu.add(newMenuItem);
experimentMenu.add(openMenuItem);
experimentMenu.add(closeMenuItem);

experimentMenu.addSeparator();

experimentMenu.add(saveMenuItem);
experimentMenu.add(saveAsMenuItem);

experimentMenu.addSeparator();

experimentMenu.add(exitMenuItem);

//-----------------
//Edit menu:
editMenu = new JMenu("Edit");


//-----------------
//Phases menu:

phasesMenu = new JMenu("Phases");

JMenuItem configurationMenuItem = new JMenuItem("Configure Experiment");
JMenuItem RTISetupMenuItem = new JMenuItem("Setup RTI Execution");
JMenuItem runMenuItem = new JMenuItem("Run Simulation");
JMenuItem abortMenuItem = new JMenuItem("Abort Simulation");
JMenuItem cleanupMenuItem = new JMenuItem("Clean Up RTI");

configurationMenuItem.setActionCommand("configure_experiment");
configurationMenuItem.addActionListener((ActionListener)parentCCenter);
RTISetupMenuItem.setActionCommand("run_rti_setup");
RTISetupMenuItem.addActionListener((ActionListener)parentCCenter);
runMenuItem.setActionCommand("run_simulation");
runMenuItem.addActionListener((ActionListener)parentCCenter);
abortMenuItem.setActionCommand("abort_simulation");
abortMenuItem.addActionListener((ActionListener)parentCCenter);
cleanupMenuItem.setActionCommand("cleanup_rti");
cleanupMenuItem.addActionListener((ActionListener)parentCCenter);

phasesMenu.add(new JLabel("Go to...."));
```

```
    phasesMenu.add(configurationMenuItem);
    phasesMenu.add(RTISetupMenuItem);
    phasesMenu.add(runMenuItem);
    phasesMenu.add(abortMenuItem);
    phasesMenu.add(cleanupMenuItem);

    //-----------------
    //put together:

    menuBar.add(experimentMenu);
    menuBar.add(editMenu);
    menuBar.add(phasesMenu);

    setJMenuBar(menuBar);


  }//end of createMenuBar()

  //=======================================================
  //=======================================================

  private void createLeftSplitPane() {

    //This Method creates and puts together the left side
    //contents of the display

    D.dbgOut("$Method CMIFControlDisplay. createLeftSplitPane()");

    //------------------
    //top part:

    upperLeftPanel =  new JPanel(new BorderLayout());


    infoPanel = new JPanel();
    infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y_AXIS));
    infoPanel.setBorder(BorderFactory.
                        createTitledBorder(lineBorder, "Info"));

    createInfoPanel();

    timePanel = new JPanel();
    timePanel.setBorder(BorderFactory.
                        createTitledBorder(lineBorder, "Time Management"));

    createTimePanel();

    //------------------
    //bottom part:

    phasesPanel = new JPanel();
    phasesPanel.setBorder(BorderFactory.
                          createTitledBorder(lineBorder, "Experiment Phases"));

    createPhasesPanel();

    //------------------
    //together:

    upperLeftPanel.add(infoPanel, BorderLayout.NORTH);
    upperLeftPanel.add(timePanel, BorderLayout.CENTER);
```

```
    leftSplitPane.setLeftComponent(upperLeftPanel);
    leftSplitPane.setRightComponent(phasesPanel);


  }//end of createLeftSplitPane

  //=====================================================
  //=====================================================

  private void createInfoPanel() {

    //This Method creates the panel that dsiplays the name of the current
    //experiment and provides a button to access the "about" dialog...

    JPanel topPanel = new JPanel();

    infoLabel = new JLabel(parentCCenter.CCManager.experimentNameString);
    infoLabel.setForeground(Color.black);

    topPanel.add(new JLabel("Name: "));
    topPanel.add(infoLabel);

    JButton aboutButton = new JButton("About...");
    aboutButton.addActionListener((ActionListener)parentCCenter);
    aboutButton.setActionCommand("display_about_text");
    aboutButton.setAlignmentX(Component.CENTER_ALIGNMENT);

    infoPanel.add(topPanel);
    infoPanel.add(aboutButton);

  }//end of createInfoPanel


  //=====================================================
  //=====================================================

  private void createTimePanel() {

    //This Method creates the panel shows disserent aspects of
    //time management and controls for it.

    D.dbgOut("$#Method CMIFControlDisplay.createTimePanel()");

    timePanel.setLayout(new BorderLayout());

    JPanel timePanel2 = new JPanel();

    timePanel2.setLayout(new GridLayout(5,2, 5, 10));

    int columns = 8;

    masterTimeTF = new JTextField("0.00", columns);
    masterTimeTF.setHorizontalAlignment(JTextField.RIGHT);
    masterTimeTF.setEnabled(false);
    masterTimeTF.setDisabledTextColor(Color.black);

    startTimeTF = new JTextField(parentCCenter.CCManager.startTimeString,
                                 columns);
    startTimeTF.setHorizontalAlignment(JTextField.RIGHT);

    endTimeTF = new JTextField(parentCCenter.CCManager.endTimeString,
```

```
                                columns);
    endTimeTF.setHorizontalAlignment(JTextField.RIGHT);

    intervalTF = new JTextField(parentCCenter.CCManager.intervalTimeString,
                                columns);
    intervalTF.setHorizontalAlignment(JTextField.RIGHT);

    scaleTF = new JTextField(parentCCenter.CCManager.scaleString,
                             columns);
    scaleTF.setHorizontalAlignment(JTextField.RIGHT);


    timePanel2.add(new JLabel("Master Time: ", JLabel.RIGHT));
    timePanel2.add(masterTimeTF);

    timePanel2.add(new JLabel("Start: ", JLabel.RIGHT));
    timePanel2.add(startTimeTF);

    timePanel2.add(new JLabel("End: ", JLabel.RIGHT));
    timePanel2.add(endTimeTF);

    timePanel2.add(new JLabel("Interval: ", JLabel.RIGHT));
    timePanel2.add(intervalTF);

    timePanel2.add(new JLabel("Scale: ", JLabel.RIGHT));
    timePanel2.add(scaleTF);


    updateButton = new JButton("Update");
    updateButton.addActionListener((ActionListener)parentCCenter.CCManager);
    updateButton.setActionCommand("update_time_management");

    timePanel.add(timePanel2, BorderLayout.CENTER);
    timePanel.add(updateButton,BorderLayout.SOUTH );


  }//end of createTimePanel()

  //=======================================================
  //=======================================================

  private void createPhasesPanel() {

    //This Method creates the panel that displays the state of the
    //simulation.

    D.dbgOut("$##Method CMIFControlDisplay.createPhasesPanel()");

    phasesPanel.setLayout(new BoxLayout(phasesPanel, BoxLayout.Y_AXIS));

    phasesInnerPanel = new JPanel();
    phasesInnerPanel.setLayout(new GridLayout(5,2,5,10));

    configLabel = new JLabel(orangeIcon);
    phasesInnerPanel.add(configLabel);
    phasesInnerPanel.add(new JLabel("Configration"));

    rtiSetupLabel= new JLabel(redIcon);
    phasesInnerPanel.add(rtiSetupLabel);
    phasesInnerPanel.add(new JLabel("RTI Setup"));

    simulationLabel = new JLabel(redIcon);
```

```
    phasesInnerPanel.add(simulationLabel);
    phasesInnerPanel.add(new JLabel("Simulation Run"));

    rtiCleanupLabel = new JLabel(redIcon);
    phasesInnerPanel.add(rtiCleanupLabel);
    phasesInnerPanel.add(new JLabel("RTI Cleanup"));

    shutdownLabel = new JLabel(orangeIcon);
    phasesInnerPanel.add(shutdownLabel);
    phasesInnerPanel.add(new JLabel("Shutdown"));

    phasesPanel.add(phasesInnerPanel);
    phasesPanel.add(Box.createVerticalGlue());


  }//end of createPhasesPanel()

  //=====================================================
  //=====================================================

  private void createRightSplitPane() {

    //This method creates and populates the right part of the
    //display window

    D.dbgOut("$Method CMIFControlDisplay.createRightSplitPane()");

    //----------------------------
    //top part:

    topRightTabbedPane = new JTabbedPane();

    participantsSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);

    //participantsSplitPane.setBorder(emptyBorder);
    participantsSplitPane.setDividerSize(1);

    channelsPanel = new JPanel();
    channelsPanel.setBorder(emptyBorder);

    communicationsPanel = new JPanel();
    communicationsPanel.setBorder(emptyBorder);

    topRightTabbedPane.addTab("Experiment Participants",
                              participantsSplitPane);
    topRightTabbedPane.addTab("Channels", channelsPanel);
    topRightTabbedPane.addTab("Communications", communicationsPanel );

    createParticipantsSplitPane();
    createChannelsPanel();
    createCommunicationsPanel();

    rightSplitPane.setLeftComponent(topRightTabbedPane);

    //----------------------------
    //bottom part:

    messagePanel = new JPanel();

    createMessagePanel();
```

```
    rightSplitPane.setRightComponent(messagePanel);


  }//end of createRightSplitPane

  //=======================================================
  //=======================================================

  private void createParticipantsSplitPane() {

    //This method creates the panel that lists
    //all participants of the current experiment, gives information
    //about their state and allows some control

    D.dbgOut("$##Method CMIFControlDisplay.createParticipantsPanel()");


    //--------------------------------
    //configuration panel

    pConfigPanel = new JPanel(new GridLayout(1,2));

    pConfigPanel.setBorder(BorderFactory.
                           createTitledBorder(lineBorder,
                                              "Participants to contact..."));
    ((GridLayout)pConfigPanel.getLayout()).setHgap(10);

    parentCCenter.CCManager.participantsConfigList.setVisibleRowCount(4);

    parentCCenter.CCManager.
      participantsConfigList.
      setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

    listScrollPane =
      new JScrollPane(parentCCenter.CCManager.participantsConfigList,
                      ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
                      ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

    listScrollPane.setBorder(emptyBorder);


    JPanel buttonPanel = new JPanel();
    //buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));
    buttonPanel.setLayout(new FlowLayout(FlowLayout.LEFT));

    addButton = new JButton("Add");
    addButton.addActionListener((ActionListener)parentCCenter.CCManager);
    addButton.setActionCommand("add_participant");

    modifyButton = new JButton("Modify");
    modifyButton.addActionListener((ActionListener)parentCCenter.CCManager);
    modifyButton.setActionCommand("modify_participant");

    deleteButton = new JButton("Delete");
    deleteButton.addActionListener((ActionListener)parentCCenter.CCManager);
    deleteButton.setActionCommand("delete_participant");

    //buttonPanel.add(Box.createVerticalGlue());
    buttonPanel.add(addButton);
    buttonPanel.add(modifyButton);
    buttonPanel.add(deleteButton);
    buttonPanel.add(Box.createVerticalGlue());
```

```
  pConfigPanel.add(listScrollPane);
  pConfigPanel.add(buttonPanel);

  //--------------------------------
  //simulation panel

  pSimulationPanel = new JPanel();

  pSimulationPanel.
    setLayout(new BoxLayout(pSimulationPanel,BoxLayout.Y_AXIS));

  pSimulationPanel.
    setBorder(BorderFactory.createTitledBorder(lineBorder,
                                                "During Simulation..."));
  pInnerPanel = new JPanel();
  pInnerPanel.setLayout(new BoxLayout(pInnerPanel, BoxLayout.Y_AXIS));


  pInnerPanel.
    add(new JLabel
        ("...will be filled once the participants "+
         "are successfully contacted!"));

  pInnerPanel.add(Box.createVerticalGlue());

  pSimulationPanel.add(pInnerPanel);

  //------------------------------

  participantsSplitPane.setLeftComponent(pConfigPanel);
  participantsSplitPane.setRightComponent(pSimulationPanel);


}//end of createParticipantsPanel()

//=====================================================
//=====================================================

private void createChannelsPanel() {

  //This Method creates the panel that lists all available communication
  //cannels, busses etc, their state and allows some control.

  D.dbgOut("$#Method CMIFControlDisplay.createChannelsPanel()");

  channelsPanel.setLayout(new GridLayout(2,1));


}//end of createChannelsPanel()


//=====================================================
//=====================================================

private void createCommunicationsPanel() {

  //This Method creates the panel that shows the interconnections of the
  //experiment participants
```

```
    D.dbgOut("$##Method CMIFControlDisplay.createCommunicationsPanel()");


  }//end of createCommunicationsPanel()

  //=======================================================
  //=======================================================

  private void createMessagePanel() {

    //This Method creates the message panel at the lower right part of
    //the window. It is used to display status messages and the like for
    //the user.

    D.dbgOut("$##Method CMIFControlDisplay.createMessagePanel()");

    messagePanel.setLayout(new BorderLayout());
    messagePanel.setBorder(BorderFactory.createEmptyBorder(8,8,8,8));

    JPanel topPanel = new JPanel(new BorderLayout());

    JButton clearMessagesButton = new JButton ("Clear");

    clearMessagesButton.setActionCommand("clear_messages");
    clearMessagesButton.addActionListener((ActionListener)parentCCenter);

    topPanel.add(new JLabel("Messages"), BorderLayout.WEST);
    topPanel.add(clearMessagesButton,BorderLayout.EAST);

    messageTextArea = new JTextArea("");

    messageTextArea.setEnabled(false);

    messageTextArea.setDisabledTextColor(Color.black);

    JScrollPane messageScrollPane =
      new JScrollPane(messageTextArea,
                      JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                      JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

    messagePanel.add(topPanel, BorderLayout.NORTH);
    messagePanel.add(messageScrollPane, BorderLayout.CENTER);


  }//end of createMessagePanel()

  //=======================================================
  //=======================================================

  private void setComponentsSizes() {

    //This method obtains the screen resolution and from there determines
    //and sets the sizes for all major components.

    D.dbgOut("$Method CMIFControlDisplay.setComponentsSizes()");

    //-------------------------
    //main window
```

```
int idealHeight = 650;
int idealWidth  = 980;

screenDimension = new Dimension(Toolkit.getDefaultToolkit().
                                getScreenSize());

D.dbgOut("$Size of screen: " + screenDimension.toString());

windowDimension = new Dimension (idealWidth,idealHeight);

if (idealHeight>screenDimension.height)
  windowDimension.height=screenDimension.height;

if (idealWidth>screenDimension.width)
  windowDimension.width=screenDimension.width;

setSize(windowDimension.width,windowDimension.height);

//-------------------------
//divider locations

int mainDividerLocation = 210;
double rightDividerLocation = 0.6;
double leftDividerLocation = 0.55;

participantsSplitPane.setDividerLocation(180);

leftSplitPane.setDividerLocation(leftDividerLocation);
rightSplitPane.setDividerLocation(rightDividerLocation);
mainSplitPane.setDividerLocation(mainDividerLocation);

//-------------------------
//other components

Dimension windowOuterDim = getSize();
Dimension windowInnerDim = new Dimension(windowOuterDim.width-10,
                                         windowOuterDim.height-30);

D.dbgOut("$#Window inner size: " + windowInnerDim.toString());


int rightWidth = mainDividerLocation;
int leftWidth = windowInnerDim.width - mainDividerLocation;
int rightTopHeight= (new Float(windowInnerDim.height *
                               rightDividerLocation)).intValue();
int leftTopHeight=(new Float(windowInnerDim.height *
                             leftDividerLocation)).intValue();
int rightBottomHeight=(new Float(windowInnerDim.height *
                                 (rightDividerLocation-1))).intValue();
int leftBottomHeight=(new Float(windowInnerDim.height *
                                (leftDividerLocation-1))).intValue();

messagePanel.setSize(new Dimension(rightWidth,
                                   rightBottomHeight));
messageTextArea.setSize(new Dimension(rightWidth,
                                      rightBottomHeight-15));
upperLeftPanel.setSize(new Dimension(leftWidth,
                                     leftTopHeight));
phasesPanel.setSize(new Dimension(leftWidth,
                                  leftBottomHeight));
```

```
    topRightTabbedPane.setSize(new Dimension(rightWidth,
                                             rightTopHeight));

    pConfigPanel.setSize(new Dimension(rightWidth-5,
                                       100));
    pConfigPanel.setSize(new Dimension(rightWidth-5,
                                       rightTopHeight-130));
    pInnerPanel.setSize(new Dimension(rightWidth-10,
                                      rightTopHeight-140));
    listScrollPane.setSize(new Dimension(250,
                                         80));

    participantsSplitPane.setDividerLocation(110);

    setLocation((new Float(screenDimension.width / 2)).intValue()
                -(new Float(windowDimension.width / 2)).intValue(),
                (new Float(screenDimension.height / 2)).intValue()
                -(new Float(windowDimension.height / 2)).intValue());
    validate();


  }//end of setComponentsSizes

  //=======================================================
  //=======================================================

  public void setPhaseIcons(String whichOne, String newState) {

    //This method allows to set the colored icons that display
    //the states of the different experiment phases.
    //The first parameter determines which phase icon will be changed,
    //the second one determines the new state.

    D.dbgOut("$Method CMIFControlDisplay.setPhaseIcons()");

    JLabel tmpLabel =null;

    if (whichOne.equals("config")) {
      tmpLabel=configLabel;

    } else if (whichOne.equals("rtisetup")) {
      tmpLabel=rtiSetupLabel;

    } else if (whichOne.equals("simulation")) {
      tmpLabel=simulationLabel;

    } else if (whichOne.equals("rticleanup")) {
      tmpLabel=rtiCleanupLabel;

    } else if (whichOne.equals("shutdown")) {
      tmpLabel=shutdownLabel;

    }


    if (tmpLabel!=null) {

      if (newState.equals("red")) {
        tmpLabel.setIcon(redIcon);
      } else if (newState.equals("red_pulse")) {
        tmpLabel.setIcon(redIconPulse);
```

```
    } else if (newState.equals("grn")) {
      tmpLabel.setIcon(greenIcon);
    } else if (newState.equals("grn_pulse")) {
      tmpLabel.setIcon(greenIconPulse);
    } else if (newState.equals("orange")) {
      tmpLabel.setIcon(orangeIcon);
    } else if (newState.equals("orange_pulse")) {
      tmpLabel.setIcon(orangeIconPulse);
    }

  }


}//end of setPhaseIcons

//=======================================================
//=======================================================

public String messageOut( String message ) {

  //This method takes a string argument and writes it to the message
  //area. Different control sequences allow to display messages to
  //the user in specific colors....as a warning for example.
  //The message itself is then returned.

  if (message.startsWith("%")) {
    //control sequence

    if (message.equals("%flush"))
      messageTextArea.setText("");

    if (message.equals("%red"))
      messageTextArea.setDisabledTextColor(Color.red);

    if (message.equals("%black"))
      messageTextArea.setDisabledTextColor(Color.black);

  } else {
    //normal message

    messageTextArea.append("\n" + message);

  }

  return message;

}//end of messageOut()

//=======================================================
//=======================================================

public Point getCenterLocation( Dimension dialogSize) {

  //This method helps to place a dialog window at the center
  //of the main display window.
  //The parameter is the size of the new dialog window and
  //the return value is the new location for the upper left
  //corner of this window to place it centered

  Rectangle currentBounds = new Rectangle (this.getBounds() );
```

```
    Point newLocation = new Point();

    newLocation.x = currentBounds.x +
      (new Float ( currentBounds.width / 2) ).intValue()-
      (new Float ( dialogSize.width / 2) ).intValue();

    newLocation.y = currentBounds.y +
      (new Float ( currentBounds.height / 2) ).intValue()-
      (new Float ( dialogSize.height / 2) ).intValue();

    return newLocation;


  } // end of getCenterLocation

  //=======================================================
  //=======================================================

  public void switchToState (int newState) {

    D.dbgOut("$Method CMIFControlDisplay.switchToState");

    //This method coordinates that all buttons and input fields
    //reflect the state of the control center and no false inputs
    //can be made

    switch (newState) {

    case 0: //idle

      break;

    case 1: //SETUP_EXPERIMENT

      startTimeTF.setEnabled(true);
      endTimeTF.setEnabled(true);
      intervalTF.setEnabled(true);
      scaleTF.setEnabled(true);
      masterTimeTF.setEnabled(true);

      updateButton.setEnabled(true);
      addButton.setEnabled(true);
      modifyButton.setEnabled(true);
      deleteButton.setEnabled(true);

      break;

    case 2: //SETUP_RTI
    case 3: //RUN_EXPERIMENT
    case 4: //CLEANUP_RTI

      startTimeTF.setEnabled(false);
      endTimeTF.setEnabled(false);
      intervalTF.setEnabled(false);
      scaleTF.setEnabled(false);
      masterTimeTF.setEnabled(false);

      updateButton.setEnabled(false);
      addButton.setEnabled(false);
      modifyButton.setEnabled(false);
      deleteButton.setEnabled(false);
```

```
      break;
    case 5:  //POST_EXPERIMENT:
      break;
    case 6: //READY_FOR_SHUTDOWN
      break;
    }//end of switch

  }//end of switchToState

  //=====================================================
  //=====================================================

  public void addParticipantControlPanel( JPanel controlPanel) {

    //This method receives the handle to a controlPanel of a
    //newly discovered experimentParticipant and puts it into place
    //in the participants simulation panel, at the top position.

    pInnerPanel.add(controlPanel, 0);
    //pInnerPanel.add(Box.createHorizontalGlue());

    pInnerPanel.revalidate();


  }//end of addParticipantControlPanel

  //=====================================================
  //=====================================================

  public void removeParticipantControlPanel( JPanel controlPanel) {

    //This method receives the handle to a controlPanel of a
    //newly discovered experimentParticipant and removes it from the
    //participants simulation panel.
    //...if the last remaining panel is removed, a note is
    //placed in the same location

    pInnerPanel.remove(controlPanel);

    pInnerPanel.revalidate();

  }//end of removeParticipantControlPanel


};//end of CMIFControlDisplay
```

D.1.3 The Class CMIFExperimentManager.java

```
//---------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                       /
//                                                         /
//This is part of a Master's Thesis at the                 /
//Center for Multisource Information Fusion                /
//SUNY at Buffalo                                          /
//                                                         /
//All rights reserved:                                     /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth          /
//packages hla.rti13.java1      : U.S. DoD                 /
//packages java, javax          : Sun Microsystems         /
//---------------------------------------------------------/

package CMIF_HLA_Environment.CMIFControlCenter;

import CMIF_HLA_Environment.util.*;
import javax.swing.JOptionPane;
import javax.swing.JFrame;

import java.awt.event.*;
import java.awt.*;
import java.util.Vector;

import javax.swing.*;

//=========================================================
//=========================================================


public class CMIFExperimentManager
  implements ActionListener

{


  //The class CMIFExperimentManager provides the control center
  //with the functionality to
  //-configure an experiment prior to the setup and usage
  // of the RTI services
  //-save experiment configurations
  //-open existing experiment configurations

  //instance variables

  public CMIFControlCenter parentCCenter;

  public DebugHelper D = new DebugHelper();


  //experiment parameters;

  public String experimentNameString = new String("UNTITLED");
  public String experimentDescriptionString = new String("n/a");
  public String experimentAuthorString =
    new String(System.getProperty("user.name"));
  public String startTimeString = new String("0.0");
  public String endTimeString = new String("100.0");
  public String intervalTimeString = new String("1.0");
```

```
public String scaleString = new String("1.0");

public double startTimeDouble = 0.0;
public double endTimeDouble = 100.0;
public double intervalTimeDouble = 1.0;
public double scaleDouble = 1.0;

public Vector participantsToJoinVector = new Vector();
public Vector channelsToEstablishVector = new Vector();


boolean timeManagementConfigured = false;
boolean participantsConfigured = false;
boolean commuicationsConfigured = false;


JDialog experimentInfoDialog;
public JTextField experimentNameTF, experimentAuthorTF;
public JTextArea experimentInfoTA;

public JList participantsConfigList = new JList(participantsToJoinVector);

//=====================================================
//=====================================================

public CMIFExperimentManager (CMIFControlCenter parent) {

  //Constructor to create an instance of CMIFExperimentManager.
  //The control center is made accessible by keeping a handle
  //(parentCenter) to it.
  //Then the experiment manager does nothing until attempts are made
  //to open a new or existing experiment.

  parentCCenter = parent;

  D.dbgOut("Started CMIFExperimentManager.....");
  D.setDebugState(DebugHelper.FULL_DEBUG);

  createExperimentInfoDialog();


}//end of constructor

//=====================================================
//=====================================================

private void createExperimentInfoDialog() {

  //The dialog window that might later be needed to ask the user
  //for name and info for an experiment is created beforehand
  //in this method

  experimentInfoDialog = new JDialog(parentCCenter.CCDisplay,
                                     "Enter Experiment Info", true);

  experimentInfoDialog.getContentPane().setLayout(new BorderLayout());

  //-----------------------------
  //upper part:

  JPanel upperPanel = new JPanel(new GridLayout(2,1));
  JPanel upperPanel1 = new JPanel(new FlowLayout(FlowLayout.LEFT));
```

```
    JPanel upperPanel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));

    experimentNameTF = new JTextField(20);
    experimentAuthorTF = new JTextField(20);
    experimentAuthorTF.setText(experimentAuthorString);

    upperPanel1.add(new JLabel("Experiment Name:  "));
    upperPanel1.add(experimentNameTF);

    upperPanel2.add(new JLabel("Experiment Author:"));
    upperPanel2.add(experimentAuthorTF);

    upperPanel.add(upperPanel1);
    upperPanel.add(upperPanel2);


    //-----------------------------
    //center part:

    JPanel centerPanel = new JPanel(new BorderLayout());

    centerPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

    experimentInfoTA = new JTextArea(10,25);

    centerPanel.add(experimentInfoTA,   BorderLayout.CENTER);

    centerPanel.add(new JLabel("Enter additional info...."), BorderLayout.NORTH);

    //-----------------------------
    //bottom part:

    JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

    JButton okButton = new JButton("Ok");
    okButton.addActionListener(this);
    okButton.setActionCommand("info_ok");

    JButton cancelButton = new JButton("Cancel");
    cancelButton.addActionListener(this);
    cancelButton.setActionCommand("info_cancel");

    bottomPanel.add(okButton);
    bottomPanel.add(cancelButton);

    experimentInfoDialog.getContentPane().add(upperPanel,BorderLayout.NORTH);
    experimentInfoDialog.getContentPane().add(centerPanel,BorderLayout.CENTER);
    experimentInfoDialog.getContentPane().add(bottomPanel,BorderLayout.SOUTH);

    experimentInfoDialog.pack();

  }//end of createExperimentInfoDialog


  //=====================================================
  //=====================================================

  public void displayAboutText() {

    //This method opens a dialog window and displays
    //the information stored in the experimentDescriptionString
```

```
    D.dbgOut("$#Method: CMIFExperimentManager.displayAboutText()");

    String tmpString = new String();

    if (experimentDescriptionString.equals("n/a")) {
      tmpString="No additional information avaliable...";
    } else {
      tmpString = experimentDescriptionString;
    }


    JOptionPane.showMessageDialog((JFrame)parentCCenter.CCDisplay ,
                                  tmpString , experimentNameString,
                                  JOptionPane.INFORMATION_MESSAGE);

  }//end of displayAboutText()

  //=====================================================
  //=====================================================

  public void configureNewExperiment() {

    //This method initiates the setup of a new experiment
    //configuration.
    //First a dialog is brought up that asks the user for
    //experiment name and description

    D.dbgOut("$#Method: CMIFExperimentManager. configureNewExperiment()");

    if (parentCCenter.cCenterState == parentCCenter.IDLE ||
        parentCCenter.cCenterState == parentCCenter.READY_FOR_SHUTDOWN) {

      parentCCenter.cCenterState= parentCCenter.SETUP_EXPERIMENT;
      parentCCenter.CCDisplay.setPhaseIcons("config","grn_pulse");

      experimentInfoDialog.
        setLocation(parentCCenter.CCDisplay.
                    getCenterLocation(experimentInfoDialog.getSize()));

      experimentInfoTA.setText("");
      experimentNameTF.setText("");


      experimentInfoDialog.show();

    } else {

      parentCCenter.CCDisplay.
        messageOut("The current state of the control center does not");
      parentCCenter.CCDisplay.
        messageOut("allow to open a new experiment!");
      parentCCenter.CCDisplay.
        messageOut("Please finish and close the current experiment first.");

    }

  }//end of configureNewExperiment

  //---------------------------------------------
  //---------------------------------------------
```

```
public void evaluateTimePanelInputs() {

  //After pressing the "update" Button on the time management panel
  //this method reads the strings from the text fields and checks for
  //valid inputs.

  D.dbgOut("$#Method: CMIFExperimentManager.evaluateTimePanelInputs()");

  boolean allOk = true;

  allOk = allOk &
    convertTimeStrings("start", parentCCenter.CCDisplay.startTimeTF.getText());

  allOk = allOk &
    convertTimeStrings("end", parentCCenter.CCDisplay.endTimeTF.getText());

  allOk = allOk &
    convertTimeStrings("interval",    parentCCenter.CCDisplay.
                       intervalTF.getText());

  allOk = allOk &
    convertTimeStrings("scale", parentCCenter.CCDisplay.scaleTF.getText());

  timeManagementConfigured=true;

  parentCCenter.CCDisplay.messageOut("");
  parentCCenter.
    CCDisplay.messageOut("Successfully updated the timing parameters.");

  if (!allOk) {
    parentCCenter.CCDisplay.messageOut("%red");
    parentCCenter.CCDisplay.messageOut("");
    parentCCenter.CCDisplay.
      messageOut("Some of the entries were not valid...please try again!");
    parentCCenter.CCDisplay.messageOut("%black");
    timeManagementConfigured=false;
  }

}//end of evaluateTimePanelInputs

//-----------------------------------------------
//-----------------------------------------------

private boolean convertTimeStrings(String whichOne, String stringTime) {

  //This method attempts to convert the string inputs made for time management
  //(either from file or from the text boxes) to the neccessare double values.
  //It returns (true/false) if the attempt was successful...

  D.dbgOut("$#Method: CMIFExperimentManager.convertTimeStrings("
           +whichOne+", " + stringTime +" )");

  boolean conversionSuccessful = false;
  double tmpDouble=0.0;

  try {

    tmpDouble = (new Double(stringTime)).doubleValue();

    D.dbgOut("$#converted to: " + tmpDouble);

    conversionSuccessful=true;
```

```
    if (whichOne.equals("start")){
      if (tmpDouble>=0) {
        startTimeDouble = tmpDouble;
        startTimeString= (new Double(startTimeDouble)).toString();
        parentCCenter.CCDisplay.startTimeTF.setText(startTimeString);
      } else {
        conversionSuccessful=false;
        parentCCenter.CCDisplay.startTimeTF.setText("");
      }
    }

    if (whichOne.equals("end")){
      if (startTimeDouble<tmpDouble) {
        endTimeDouble = tmpDouble;
        endTimeString= (new Double(endTimeDouble)).toString();
        parentCCenter.CCDisplay.endTimeTF.setText(endTimeString);
      } else {
        conversionSuccessful=false;
        parentCCenter.CCDisplay.endTimeTF.setText("");
      }
    }

    if (whichOne.equals("interval")) {
      if (tmpDouble<(endTimeDouble-startTimeDouble)/2) {
        intervalTimeDouble = tmpDouble;
        intervalTimeString= (new Double(intervalTimeDouble)).toString();
        parentCCenter.CCDisplay.intervalTF.setText(intervalTimeString);
      } else {
        conversionSuccessful=false;
        parentCCenter.CCDisplay.intervalTF.setText("");
      }
    }

    if (whichOne.equals("scale") ){
      if (tmpDouble>0) {
        scaleDouble = tmpDouble;
        scaleString= (new Double(scaleDouble)).toString();
        parentCCenter.CCDisplay.scaleTF.setText(scaleString);
      } else{
        conversionSuccessful=false;
        parentCCenter.CCDisplay.scaleTF.setText("");
      }
    }


  } catch (Exception e){

    D.dbgOut("Error in convertTimeStrings: " + e.toString());
    conversionSuccessful=false;
    if (whichOne.equals("start")) {
      parentCCenter.CCDisplay.startTimeTF.setText("");
    } else if (whichOne.equals("end")) {
      parentCCenter.CCDisplay.endTimeTF.setText("");
    } else if (whichOne.equals("scale")) {
      parentCCenter.CCDisplay.scaleTF.setText("");
    } else if (whichOne.equals("interval")){
      parentCCenter.CCDisplay.intervalTF.setText("");
    }
```

```
    }

    return conversionSuccessful;

  }//end of convertTimeStrings


  //-----------------------------------------
  //-----------------------------------------

  public boolean configComplete() {

    //This method allows other program parts to check on the
    //status of the experiment configuration.
    //It returns true only if all of the following are defined;
    //time management, participants to contact, communications.

    D.dbgOut("$#Method: CMIFExperimentManager.configComplete()");

    boolean result = false;

    participantsConfigured = participantsToJoinVector.size()>0;

                                //for the time being:
    commuicationsConfigured = true;


    if ((timeManagementConfigured && participantsConfigured) &&
        commuicationsConfigured)
      result = true;

    return result;
    //return true;


  }//end of configComplete

  //-----------------------------------------
  //-----------------------------------------

  public void editParticipantsToJoinList(String operationString) {

    //This method processes attempts to change the content of the
    //lists of experiment participants in the participants panel.
    //The parameter operationString contains on  of "add", "modify"
    //and "delete" and determines what should be done.
    //All this can only be processed durin the configuration phase...

    D.dbgOut("Method ExperimentManager.editParticipantsToJoinList" +
             "(String operationString)");

    if (parentCCenter.cCenterState == parentCCenter.SETUP_EXPERIMENT) {

      if (operationString.equals("add")) {

        String message = new String("Add a new experiment participant");

        String newParticipantString =
          JOptionPane.showInputDialog(parentCCenter.CCDisplay,
                                      message,
                                      "Participants List",
                                      JOptionPane.QUESTION_MESSAGE);
```

```
    if (!participantsToJoinVector.contains(newParticipantString)) {

      participantsToJoinVector.add(newParticipantString);
      participantsConfigList.setListData(participantsToJoinVector);

      participantsConfigList.validate();

    } else {

      parentCCenter.CCDisplay.messageOut("");
      parentCCenter.
        CCDisplay.messageOut("Participants with identical names" +
                             " are not allowed.");
    }


  } else if (operationString.equals("modify")) {

    if (participantsConfigList.getSelectedIndex()!=-1) {

      String message = new String("Modify this experiment participant");

      String selectedParticipantString = (String)participantsConfigList.
        getSelectedValue();

      String newParticipantString =
        (String)JOptionPane.showInputDialog(parentCCenter.CCDisplay,
                                            message,
                                            "Participants List",
                                            JOptionPane.QUESTION_MESSAGE,
                                            null,
                                            null,
                                            selectedParticipantString);

      if (!participantsToJoinVector.contains(newParticipantString)) {

        participantsToJoinVector.add(newParticipantString);
        participantsToJoinVector.remove(selectedParticipantString);

        participantsConfigList.setListData(participantsToJoinVector);

        participantsConfigList.validate();

      } else {

        parentCCenter.CCDisplay.messageOut("");
        parentCCenter.
          CCDisplay.messageOut("Participants with identical names" +
                               " are not allowed. No changes made...");

      }

    }

  } else if (operationString.equals("delete")) {

    if (participantsConfigList.getSelectedIndex()!=-1) {

      String selectedParticipantString = (String)participantsConfigList.
        getSelectedValue();
```

```
        participantsToJoinVector.remove(selectedParticipantString);
        participantsConfigList.setListData(participantsToJoinVector);

        participantsConfigList.validate();

      }

    }

  } else {

    parentCCenter.CCDisplay.messageOut("");
    parentCCenter.
      CCDisplay.messageOut("The current phase does not allow to change" +
                           " the contents of this list.");

  }

}//end of editParticipantsToJoinList


//-----------------------------------------------
//-----------------------------------------------

// Listener

//-----------------------------------------------

public void actionPerformed(ActionEvent event) {

  String command = event.getActionCommand();

  D.dbgOut("ExperimentManager.actionPerformed() received: " + command);

  if (command.equals("info_ok")){

    experimentNameString = experimentNameTF.getText();
    experimentAuthorString = experimentAuthorTF.getText();
    experimentDescriptionString = experimentInfoTA.getText();

    parentCCenter.CCDisplay.infoLabel.setText(experimentNameString);

    experimentInfoDialog.setVisible(false);

  }

  //------------------------------

  if (command.equals("info_cancel")){

    experimentInfoDialog.setVisible(false);

  }

  //------------------------------

  if (command.equals("update_time_management")){

    if (parentCCenter.cCenterState==parentCCenter.SETUP_EXPERIMENT)
      evaluateTimePanelInputs();

  }
```

```
    //------------------------------

    if (command.equals("add_participant"))
      editParticipantsToJoinList("add");

    if (command.equals("modify_participant"))
      editParticipantsToJoinList("modify");

    if (command.equals("delete_participant"))
      editParticipantsToJoinList("delete");

  }//end of actionPerformed


};//end of class CMIFExperimentManager
```

D.1.4 The Class RTIControlModule.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                  : CMIF, Kai Harth          /
//packages hla.rti13.java1    : U.S. DoD                 /
//packages java, javax        : Sun Microsystems         /
//-------------------------------------------------------/

package CMIF_HLA_Environment.CMIFControlCenter;


import CMIF_HLA_Environment.util.*;
import CMIF_HLA_Environment.CMIFExperimentParticipant.*;

import hla.rti13.java1.*;

import java.awt.event.*;
import java.util.Date;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.StringTokenizer;

//=========================================================
//=========================================================

public class RTIControlModule
  implements ActionListener

{

  //The class RTIControlModule represents the HLA object of the same name.
  //It sets up and manages all RTI related functionality for the
  //CMIF Control Center.

  //static variables:

  public static int RTI_DOWN=1;
  public static int RTI_SETUP=2;
  public static int RTI_RUNNING=3;
  public static int RTI_SHUTDOWN=4;
  public static int RTI_CORRUPTED=5;

  public static int SIM_IDLE=6;
  public static int SIM_READY=7;
  public static int SIM_RUNNING=8;
  public static int SIM_DONE=9;
  public static int SIM_CORRUPTED=9;
  public static int SIM_ABORTED=10;

  //handles

  static int hRTIControlModule, hExperimentState, hExperimentParticipant,
    hParticipantName, hParticipantState, hControlInteraction,
```

```
    hControlMessageContent, hControlMessageRecipient, hMessageToBus,
    hToBusMessageContent, hToBusSender, hToBusRecipient, hMessageFromBus,
    hFromBusMessageContent, hFromBusSender, hFromBusRecipient, hStatusMessage,
    hStatusMessageContent, hStatusMessageSender;

  /* keep a reference to the RTIambassador for use from within TestNode
     methods */

  protected static RTIambassador sRtiAmb;

  //instance variables:

  //the timing values are set after those in the
  //experiment manager ...only for convenience

  public double startTimeDouble = 0.0;
  public double endTimeDouble = 100.0;
  public double intervalTimeDouble = 1.0;
  public double scaleDouble = 1.0;

  DebugHelper D = new DebugHelper ();

  CMIFControlCenter parentCCenter;

  public RTIambassador rtiamb;

  RTIControlModuleFedamb fedamb;

  String fedname = "RTIControlModule";

  int instanceID;

  public int rtiState = RTI_DOWN;

  public int simState = SIM_IDLE;

  public boolean issueUpdates = false;
  public boolean runTickThread = true;
  public boolean shutdownInitiated = false;

  public Hashtable remoteParticipantsTable = new Hashtable(33);


  //----------------------------------------------------------------
  //----------------------------------------------------------------
  /****************** Static Methods *********************************/

  public static void initialize(RTIambassador rtiamb) throws RTIexception {

    //This method initializes all object and attribute handles with the rtiAmb
    //...necessary to start or join a federation execution


    // hold onto a reference to our RTI ambassador

    sRtiAmb = rtiamb;

    // obtain RTI handles to the FED entities used in the federation

    hRTIControlModule = rtiamb.getObjectClassHandle("RTIControlModule");
    hExperimentState =  rtiamb.getAttributeHandle("ExperimentState",
                                                  hRTIControlModule);
```

```
    hExperimentParticipant =
      rtiamb.getObjectClassHandle("CMIFExperimentParticipant");
    hParticipantName =  rtiamb.getAttributeHandle("ParticipantName",
                                                  hExperimentParticipant);
    hParticipantState =  rtiamb.getAttributeHandle("ParticipantState",
                                                   hExperimentParticipant);


    hControlInteraction = rtiamb.getInteractionClassHandle("ControlInteraction");
    hControlMessageContent =
      rtiamb.getParameterHandle("ControlMessageContent", hControlInteraction);
    hControlMessageRecipient =
      rtiamb.getParameterHandle("ControlMessageRecipient", hControlInteraction);

    hMessageToBus= rtiamb.getInteractionClassHandle("MessageToBus");
    hToBusMessageContent=
      rtiamb.getParameterHandle("ToBusMessageContent",hMessageToBus );
    hToBusSender=
      rtiamb.getParameterHandle("ToBusSender",hMessageToBus );
    hToBusRecipient=
      rtiamb.getParameterHandle("ToBusRecipient",hMessageToBus );


    hMessageFromBus= rtiamb.getInteractionClassHandle("MessageFromBus");
    hFromBusMessageContent=
      rtiamb.getParameterHandle("FromBusMessageContent",hMessageFromBus );
    hFromBusSender=
      rtiamb.getParameterHandle("FromBusSender",hMessageFromBus );
    hFromBusRecipient=
      rtiamb.getParameterHandle("FromBusRecipient",hMessageFromBus );


    hStatusMessage= rtiamb.getInteractionClassHandle("StatusMessage");
    hStatusMessageContent=
      rtiamb.getParameterHandle("StatusMessageContent",hStatusMessage );
    hStatusMessageSender=
      rtiamb.getParameterHandle("StatusMessageSender",hStatusMessage );

  }//end of initialize

  //----------------------------------------------------------------
  //----------------------------------------------------------------

  public static void publishAndSubscribe() throws RTIexception {

    //PUBLISHING:
    //create an outgoing attribute-handle set

    AttributeHandleSet ahsetOut = AttributeHandleSetFactory.create(1);

    // populate the set with the attributes we publish/subscribe
    ahsetOut.add(hExperimentState);

    // perform the appropriate DM calls
    sRtiAmb.publishObjectClass(hRTIControlModule, ahsetOut);

    sRtiAmb.publishInteractionClass(hControlInteraction);
    sRtiAmb.publishInteractionClass(hMessageFromBus);
```

```
    //SUBSCRIBING:
    //create an attribute-handle set for experiment participant

    AttributeHandleSet ahsetIn = AttributeHandleSetFactory.create(2);

    ahsetIn.add(hParticipantName);
    ahsetIn.add(hParticipantState);

    // perform the appropriate DM calls

    sRtiAmb.subscribeObjectClassAttributes(hExperimentParticipant, ahsetIn);

    sRtiAmb.subscribeInteractionClass(hMessageToBus);
    sRtiAmb.subscribeInteractionClass(hStatusMessage);

  }//end of publishAndSubscribe

  //----------------------------------------------------------------
  //----------------------------------------------------------------

  public void receiveStatusMessage(ReceivedInteraction phvpset,
                                   String tag) {

    //This method is called when the Federate Ambassador receives
    //any status message.
    //Then the content has to be processed.

    D.dbgOut(fedname + " received StatusMessage, " + tag);

    try {


      //code to react to status message (e.g. log it or display it
      //will go here in next program version

    } catch (Exception e){

      D.dbgOut("Exception in RTIControlModule."+
               "receiveStatusMessage:" );
      D.dbgOut("#" + e.toString());

    }


  }//end of receiveStatusMessage

  //=========================================================
  //=========================================================

  public void receiveMessageToBus(ReceivedInteraction phvpset,
                                  String tag) {

    //This method is called when the Federate Ambassador receives
    //any message from bus.
    //Then it has to be evaluated, which recipient the message is intended for
    //and the content has to be processed.

    D.dbgOut(fedname + " received MessageToBus, " + tag);

    try {

      String messageSender =
```

```
        EncodingHelpers.decodeString(phvpset.getValue(2));

      String messageRecipient =
        EncodingHelpers.decodeString(phvpset.getValue(1));

      String message =
        EncodingHelpers.decodeString(phvpset.getValue(0));

      D.dbgOut("$#....message: " + message + " for " + messageRecipient);

      if (messageRecipient.equals("all")) {

        Enumeration remoteParticipantsEnum = remoteParticipantsTable.elements();

        while (remoteParticipantsEnum.hasMoreElements()){

          CMIFExperimentParticipant tmpParticipant =
            (CMIFExperimentParticipant)remoteParticipantsEnum.nextElement();

          if (sendingIsAllowed(messageSender, tmpParticipant.participantName)){

            sendMessageToBus(messageSender,
                             tmpParticipant.participantName,
                             message);

          }

        }

      } else {

        if (sendingIsAllowed(messageSender, messageRecipient)){

          sendMessageToBus(messageSender, messageRecipient, message);

        }

      }

      //Other things to do with an incoming bus message
      //will go here in a future program version...

      //logMessage();
      //evaluateDataVolume();


    } catch (Exception e){

      D.dbgOut("Exception in RTIControlModule."+
               "receiveMessageToBus:" );
      D.dbgOut("#" + e.toString());

    }

  }//end of receiveMessageToBus

  //============================================================
  //============================================================

  public boolean sendingIsAllowed(String sender, String recipient) {
```

```
  //This method evaluates the communication matrix (later to be implemented)
  //and returns true if "sender" is currently allowed to send to
  //"recipient"

  return (!sender.equals(recipient));

}//end of sendingIsAllowed



//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void sendControlInteraction(String recipient,
                                   String message)  {

  //This method is used to send a control interaction,
  //represented by the string 'message' to the recipient.
  //If 'all' is used as recipient, all participants
  //will accept the message.

  try {

    D.dbgOut("Sending ControlInteraction...");
    D.dbgOut("#Recipient: " + recipient + ", Content: " +message );


    // create the outgoing parameters set
    SuppliedParameters phvpset = SuppliedParametersFactory.create(2);


    // add the bytes to the parameter set
    phvpset.add(hControlMessageRecipient,
                EncodingHelpers.encodeString(recipient));
    phvpset.add(hControlMessageContent,
                EncodingHelpers.encodeString(message));

    // send the interaction on its way
    sRtiAmb.sendInteraction(hControlInteraction, phvpset, "Java");

  } catch (Exception e) {

    D.dbgOut("Exception in RTIControlModule."+
             "sendControlInteraction:" );
    D.dbgOut("#" + e.toString());

  }

}//end of sendControlInteraction

//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void sendMessageToBus(String sender,
                             String recipient,
                             String message)  {

  //This method is used to forward any message, coming from a
  //participant, back to the bus
```

```
    //and to the recipient.

    try {

      D.dbgOut("Sending MessageToBus...");
      D.dbgOut("#Sender:" + sender + ", Recipient: " + recipient
               + ", Content: " +message );

      // create the outgoing parameters set
      SuppliedParameters phvpset = SuppliedParametersFactory.create(3);

      // add the bytes to the parameter set
      phvpset.add(hFromBusSender,
                  EncodingHelpers.encodeString(sender));
      phvpset.add(hFromBusRecipient,
                  EncodingHelpers.encodeString(recipient));
      phvpset.add(hFromBusMessageContent,
                  EncodingHelpers.encodeString(message));

      // send the interaction on its way
      sRtiAmb.sendInteraction(hMessageFromBus, phvpset, "Java");

    } catch (Exception e) {

      D.dbgOut("Exception in RTIControlModule."+
               "sendMessageToBus:" );
      D.dbgOut("#" + e.toString());

    }

  }//end of sendMessageToBus

  //=====================================================
  //=====================================================
  //Per instance methods
  //=====================================================
  //=====================================================

  public RTIControlModule (CMIFControlCenter parent) {

    //Constructor for the class RTIControlModule
    //Parameter is the handle to the Controlcenter....from
    //there, all other parts of the program are accessible.

    D.dbgOut("Started RTIControlModule.....");
    D.setDebugState(DebugHelper.FULL_DEBUG);

    parentCCenter = parent;

    runBackgroundTickThread();

  }//end of contructor

  //=====================================================
  //=====================================================

  public void launchRTI() {

    //This method kicks off the RTI execution

    rtiState=RTI_SETUP;
```

```
D.dbgOut("Method: RTIControlModule.launchRTI() ");

fedamb = new RTIControlModuleFedamb(this);

try {

  // construct the RTI ambassador
  rtiamb = new RTIambassador();

  try {
    // attempt to create the federation executive
    rtiamb.createFederationExecution("CMIF_HLA_Environment",
                                     "CMIF_HLA_Environment.fed");
    D.dbgOut(fedname + ": created federation execution");

    rtiState=RTI_RUNNING;

  } catch (FederationExecutionAlreadyExists ex) {
    D.dbgOut(fedname + ": federation execution already exists");
    // this exception is okay, so fall through
  }

  /* The "fedex" sets itself up asynchronously with respect to the
     return of the "createFederationExecution" call, so join attempts
     occuring quickly afterwards will fail. If this happens, we'll keep
     looping and trying.
  */

  int tries = 30;
  while (tries > 0) {
    try {
      rtiamb.joinFederationExecution(fedname,
                                     "CMIF_HLA_Environment",
                                     fedamb);
      break;
    }
    catch (FederationExecutionDoesNotExist ex) {
      D.dbgOut(fedname + ": federation does not exist");

      // decrement the number of remaining tries and
      //tick to kill some time

      tries--;
      rtiamb.tick(0.4, 0.5);
    }
  }

  /* If we still haven't joined, try once more and let the exception
     propagate. */

  if (tries == 0)
    rtiamb.joinFederationExecution(fedname,
                                   "CMIF_HLA_Environment",
                                   fedamb);

  D.dbgOut(fedname + ": joined federation execution");

  rtiState=RTI_RUNNING;

} catch (RTIexception ex) {

  /* catch exceptions that result from creating the RTI ambassador,
```

```
      creating the federation executive, or joining the federation
      execution */

    D.dbgOut(fedname + ": caught exception " + ex);

    rtiState=RTI_CORRUPTED;

    parentCCenter.CCDisplay.messageOut("");
    parentCCenter.
      CCDisplay.messageOut("An Error occured...the RTI could not "+
                           "be launched!");

    return;
  }

  try {

    initialize(rtiamb);
    D.dbgOut(fedname + ": established RTI<->SIM handle mapping");

    publishAndSubscribe();
    D.dbgOut(fedname + ": initialized DM disposition");

    instanceID = sRtiAmb.registerObjectInstance(hRTIControlModule);

    // enable time-constraint

    rtiamb.enableTimeConstrained();

    /* our federate ambassador will set a boolean variable to "true" when
       it receives notification that constraint has been enabled; we'll
       loop and tick until this happens
    */

    while (!fedamb.mConstraintEnabled) {
      rtiamb.tick(0.01, 0.2);
    }
    D.dbgOut(fedname + ": time constraint enabled @ " +
             fedamb.mCurrentTime);

    // enble time regulation at the current time and lookahead
    rtiamb.enableTimeRegulation( EncodingHelpers.encodeDouble
                                 ( fedamb.mCurrentTime ),
                                 rtiamb.queryLookahead() );
    /* loop until the federate ambassador indicates that regulation has been
       enabled */

    while (!fedamb.mRegulationEnabled) {
      rtiamb.tick(0.01, 0.2);
    }

    D.dbgOut(fedname + ": time regulation enabled @ "+
             fedamb.mCurrentTime);

    // enable asynchronous delivery of receive-ordered events

    rtiamb.enableAsynchronousDelivery();
    D.dbgOut(fedname + ": asynchronous delivery enabled");

    rtiamb.tick();

  } catch (Exception e) {
```

```
    D.dbgOut("Exception occured in runPreSimulation: " + e.getMessage());
  }

}//end of launchRTI

//=====================================================
//=====================================================

public void runPreSimulation() {

  //This method kicks off initialization of the HLA representation of the
  //RTIControlModule, object publication and subscription and then tries
  //to contact the experiment participants

  D.dbgOut("$Method: RTIControlModule.runPreSimuation()");

  try {

    while(!issueUpdates)
      rtiamb.tick(0.2, 0.4);

    updateAttributeValues(fedamb.mCurrentTime);

    //wait for all participants to be located

    while(!allParticipantsLocated() ) {

      rtiamb.tick();
      if (rtiState==RTI_SHUTDOWN){
        shutDownRTIinOwnThread();
        break;}
    }

    D.dbgOut("RTIControlModule has located all expected participants!");
    D.dbgOut("#...now set them up for the experiment....");

    //send them the timing info

    sendOutTimingInfo();

    //wait for them to be ready for the simulation

    while(!allParticipantsReadyForSimulation() ) {

      rtiamb.tick();
      if (rtiState==RTI_SHUTDOWN){
        shutDownRTIinOwnThread();
        break;}
    }

    D.dbgOut("All Participants report ready for simulation!");

    simState=SIM_READY;

  } catch (Exception e) {

    D.dbgOut("Exception in RTIControlModule.runPreSimulation() "+
             e.getMessage());

  }
```

```
}//end of runPreSimulation

//=====================================================
//=====================================================

public void runSimulation() {

  //This method kicks of the actual simulation run
  //First, a control message is sent out to the
  //participants, then the master time is provided.

  D.dbgOut("$Method: RTIControlModule.startSimuation()");

  simState=SIM_RUNNING;

  sendControlInteraction("all", "start_simulation");

  final SwingWorker worker = new SwingWorker() {
    public Object construct() {
      //background code goes here

      D.dbgOut("RTIControlModule Starting Simulation thread!");

      try {

        //-------------------------------------------------
        //set fedamb time to desired simulation start time

        fedamb.mTimeAdvanceGrant=false;

        rtiamb.timeAdvanceRequest( EncodingHelpers.encodeDouble
                                   (startTimeDouble) );

        // loop and tick until we receive the time advance

        while (!fedamb.mTimeAdvanceGrant) {
          rtiamb.tick(0.01, 0.3);
        }
        D.dbgOut("$% RTIControlModule: Set fedamb to start time");

        updateSimTime(fedamb.mCurrentTime);


        //-----------------------------------------
        //get current system time

        long startTime = (new Date()).getTime();

        long currentTime;

        int iterations = 1;

        parentCCenter.CCDisplay.messageOut("");
        parentCCenter.CCDisplay.messageOut("Simulation is running...");

        while (simState==SIM_RUNNING) {

          currentTime = (new Date()).getTime();

          while (currentTime<= startTime +
                 (new Double(iterations *
```

```
                                intervalTimeDouble*1000/scaleDouble)).intValue()){
            currentTime = (new Date()).getTime();
          }

          double newFedambTime =
            startTimeDouble+(iterations*intervalTimeDouble);

          fedamb.mTimeAdvanceGrant=false;

          rtiamb.timeAdvanceRequest( EncodingHelpers.encodeDouble
                                     (newFedambTime ));

          // loop and tick until we receive the time advance

          while (!fedamb.mTimeAdvanceGrant) {
            rtiamb.tick(0.01, 0.3);
          }

          D.dbgOut("$% RTIControlModule: Set fedamb to time "
                   +fedamb.mCurrentTime);

          updateSimTime(fedamb.mCurrentTime);

          if (fedamb.mCurrentTime>=endTimeDouble){

            simState=SIM_DONE;
            D.dbgOut("$% RTIControlModule:Simulation is done");

          }
          iterations++;

        }//end of while

        parentCCenter.CCDisplay.messageOut("");
        parentCCenter.CCDisplay.messageOut("Simulation is completed...");

        parentCCenter.CCDisplay.setPhaseIcons("simulation", "red");
        parentCCenter.CCDisplay.setPhaseIcons("rticleanup", "orange");

      } catch (Exception e) {

        D.dbgOut("RTIControlModule: Exception in Simulation thread!");
        D.dbgOut("% " + e.toString());

        simState=SIM_CORRUPTED;

      }

      D.dbgOut("RTIControlModule: Stopped Simulation thread!");
      return null;
    }
  };

  worker.start();

}//end of runSimulation

//=================================================
//=================================================
```

```
private void updateSimTime(double simTime) {

  //this method adjusts the time display in the main window to
  //the current simulation time

  parentCCenter.CCDisplay.masterTimeTF.
    setText((new Double(simTime)).toString());
  parentCCenter.CCDisplay.masterTimeTF.revalidate();

}//end of updateSimTime

//=========================================================
//=========================================================

public void abortSimulation() {

  //This method interrupts the simulation run....
  //First, a control message is sent out to the
  //participants, then the simulation thread is stopped
  //by changing the simState.

  D.dbgOut("$Method: RTIControlModule.abortSimuation()");

  simState=SIM_ABORTED;

  sendControlInteraction("all", "abort_simulation");

}//end of abortSimulation

//=========================================================
//=========================================================

private void sendOutTimingInfo() {

  //This method encodes the timing settings of the experiment
  //(start, end, scale, interval) into a string and sends it to all
  //participants via a controlInteraction

  StringBuffer timingInfoStringBuffer =
    new StringBuffer("timing#");

  timingInfoStringBuffer.append("start#");
  timingInfoStringBuffer.
    append(parentCCenter.CCManager.startTimeString);
  timingInfoStringBuffer.append("#end#");
  timingInfoStringBuffer.
    append(parentCCenter.CCManager.endTimeString);
  timingInfoStringBuffer.append("#interval#");
  timingInfoStringBuffer.
    append(parentCCenter.CCManager.intervalTimeString);
  timingInfoStringBuffer.append("#scale#");
  timingInfoStringBuffer.
    append(parentCCenter.CCManager.scaleString);

  sendControlInteraction("all", timingInfoStringBuffer.toString());

  //also the timing values are synchronized with those in the experimentmanager
  //...just for convenience of shorter variable calls

  startTimeDouble =parentCCenter.CCManager.startTimeDouble;
```

```
    endTimeDouble = parentCCenter.CCManager.endTimeDouble;
    intervalTimeDouble = parentCCenter.CCManager.intervalTimeDouble;
    scaleDouble = parentCCenter.CCManager.scaleDouble;

  }//end of sendOutTimingInfo

  //=====================================================
  //=====================================================

  private boolean allParticipantsReadyForSimulation() {

    //This method returns false until all located participants
    //have reported ready for simulation

    boolean allReady = true;

    Enumeration remoteParticipantsEnum = remoteParticipantsTable.elements();

    while (remoteParticipantsEnum.hasMoreElements()){

      CMIFExperimentParticipant tmpParticipant =
        (CMIFExperimentParticipant)remoteParticipantsEnum.nextElement();

      D.dbgOut("$# State " + tmpParticipant.participantState );

      allReady = allReady && tmpParticipant.participantState==37;

    }

    return allReady;

  }//end of allParticipantsReadyForSimulation

  //=====================================================
  //=====================================================

  private boolean allParticipantsLocated() {

    //This method compares the names of the participants
    //in the remoteParticipantsTable with those listed in
    //participantsToJoinVector

    //D.dbgOut("Method: RTIControlModule.allParticipantsLocated");

    boolean allLocated = false;

    int alreadyFound =0;

    Enumeration remoteParticipantsEnum = remoteParticipantsTable.elements();

    while (remoteParticipantsEnum.hasMoreElements()){

      CMIFExperimentParticipant tmpParticipant =
        (CMIFExperimentParticipant)remoteParticipantsEnum.nextElement();

      if (parentCCenter.CCManager.
          participantsToJoinVector.
          contains((String)tmpParticipant.participantName)) {

        alreadyFound++;
        //D.dbgOut("$#....located: " + tmpParticipant.participantName);
      }
```

```
    }

    if (alreadyFound==parentCCenter.CCManager.
        participantsToJoinVector.size())
      allLocated = true;

    D.dbgOut("$#Remote participants: " +
             remoteParticipantsTable.size() + ", allLocated: " +allLocated);

    return allLocated;

  }//end of allParticipantsLocated

  //-----------------------------------------------------------------
  //-----------------------------------------------------------------

  public void reflectAttributeValues(int oid,
                                     ReflectedAttributes ahvpset,
                                     String theTag) throws ObjectNotKnown,
                                     AttributeNotKnown,
                                     FederateInternalError {

    //This method is triggered from the Fedamb and reacts to a callback
    //due to a change of attributes in one of the object classes that we
    //are registered for.
    //In particular, the local instance representing this object has
    //to be retrieved an its corresponding attributes have to be changed.

    D.dbgOut("Method: RTIControlModuleFedamb.reflectAttributeValues()");

    // look up the object ID in our remote instance hash table

    CMIFExperimentParticipant tmpInstance =
      (CMIFExperimentParticipant)remoteParticipantsTable.
      get(new Integer(oid));

    /* if "get" returns null, the ID was not found, so throw an
       exception
    */

    if (tmpInstance == null)
      throw new ObjectNotKnown("Object not found " + oid);

    tmpInstance.reflectAttributeValues(ahvpset);


  }//end of reflectAttributeValues

  //=================================================
  //=================================================

  private void updateAttributeValues(double theTime) {

    //This method packs the information about the controlCenter
    //(currently only the experimentState) into an AttributeHandleSet
    //and publishes it.

    if (issueUpdates) {

      try {

        // create a set for the outgoing attributes
```

```
      SuppliedAttributes ahvpset = SuppliedAttributesFactory.create(1);

      ahvpset.add(hExperimentState,
                  EncodingHelpers.encodeInt(simState + rtiState*10));

      // send the update on its way
      sRtiAmb.updateAttributeValues(instanceID,
                                    ahvpset,
                                    //EncodingHelpers.encodeDouble(theTime),
                                    "Java" );

    } catch (Exception e) {

      D.dbgOut("Exception in RTIControlModule.updateAttributeValues() " +
               e.getMessage());

    }

  }

}//end of updateAttributeValues

//=====================================================
//=====================================================
//=====================================================
//=====================================================

public void shutDownRTIinOwnThread() {

  final SwingWorker worker = new SwingWorker() {
    public Object construct() {
      //...code that might take a while
      //to execute is here...

      sendControlInteraction("all","shutdown");
      shutDownRTI();
      parentCCenter.CCDisplay.setPhaseIcons("rticleanup", "red");
      parentCCenter.CCDisplay.setPhaseIcons("shutdown", "orange");
      return null;
    }
  };
  worker.start();

}//end of shutDownRTIinOwnThread

//=====================================================
//=====================================================

public void shutDownRTI() {

  //This method tries to resign from the federation execution
  //and destroy it....

  D.dbgOut("Method:RTIControlModule.shutDownRTI");

  if ((rtiState==RTI_SHUTDOWN) && simState!=SIM_RUNNING) {

    try {
      // try to resign

      rtiamb.resignFederationExecution
```

```
        (ResignAction.DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES);
      D.dbgOut(fedname + ": resigned from federation");

      rtiState=RTI_DOWN;
    }

    catch (RTIexception ex) {
      D.dbgOut(fedname + ": caught exception while resigning " + ex);
      rtiState=RTI_CORRUPTED;
      return;
    }

    int tries =1;

    while (tries<10) {

      try {

        // try to destory the federation
        rtiamb.destroyFederationExecution("CMIF_HLA_Environment");
        D.dbgOut(fedname + ": destroyed federation execution");

        rtiState=RTI_DOWN;

      }
      catch (FederatesCurrentlyJoined ex) {
        // this exception is okay

        System.err.println
          (fedname +
           ":: federates still joined -- not destorying fedex");
      }

      catch (RTIexception ex) {

        rtiState=RTI_CORRUPTED;

        System.err.println
          (fedname +
           ": caught exception while destroying fedex " + ex);
      }

      tries++;

    }

  } else {


  }

}//end of shutDownRTI


//=====================================================
//=====================================================

private void runBackgroundTickThread() {

  //This method is started in the beginning and ensures, that whenever
  //no other phase is using (and thus ticking) the RTI, ticks are sent.
```

```
    final SwingWorker worker = new SwingWorker() {
      public Object construct() {
        //background code goes here

        D.dbgOut("Starting background tick thread!");

        while (runTickThread) {

          if (rtiState==RTI_SHUTDOWN && !shutdownInitiated){
            shutdownInitiated = true;
            shutDownRTIinOwnThread();
          }

          if (rtiState==RTI_RUNNING) {

            if (simState==SIM_READY || simState==SIM_DONE ||
                simState==SIM_ABORTED  ||simState==SIM_CORRUPTED) {

              try {
                rtiamb.tick();
              } catch (Exception e) {

                D.dbgOut("Exception in background tick thread:");
                D.dbgOut("...:" + e.getMessage());

              }
            }
          }

        }//end of while

        D.dbgOut("Stopped background tick thread!");
        return null;
      }
    };
    worker.start(Thread.MIN_PRIORITY);

  }//end of runBackgroundTickThread

  //=====================================================
  //=====================================================
  //
  // L I S T E N E R
  //
  //=====================================================

  public void actionPerformed(ActionEvent event) {

    String command = event.getActionCommand();

    D.dbgOut("RTIControlModule.actionPerformed() received: " + command);

    String delim = "@";

    if (command.indexOf(delim)!=-1){

      StringTokenizer commandTokenizer =
        new StringTokenizer(command, delim);

      String operation = commandTokenizer.nextToken();
      String issuedBy = commandTokenizer.nextToken();
```

```
      D.dbgOut("#.... operation: " + operation +", isuued by: " +issuedBy );
      sendControlInteraction(issuedBy, operation);
    }
    //-------------------------------
  }//end of actionPerformed

};//end of RTIControlModule
```

D.1.5 The Class RTIControlModuleFedamb.java

```
//-----------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                         /
//                                                           /
//This is part of a Master's Thesis at the                   /
//Center for Multisource Information Fusion                  /
//SUNY at Buffalo                                            /
//                                                           /
//All rights reserved:                                       /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth            /
//packages hla.rti13.java1      : U.S. DoD                   /
//packages java, javax          : Sun Microsystems           /
//-----------------------------------------------------------/

package CMIF_HLA_Environment.CMIFControlCenter;

import CMIF_HLA_Environment.CMIFExperimentParticipant.*;
import CMIF_HLA_Environment.util.*;

// import the RTI classes into our namespace
import hla.rti13.java1.*;

import java.util.Hashtable;

/* we only implement a small subset of FederateAmbassador methods, so derive
   our ambassador from NullFederateAmbassador to pick up no-op implementations
   of the others
*/

class RTIControlModuleFedamb extends NullFederateAmbassador {


  /* these variables are checked by the main loop to detect when regulation
     and constraint have been enabled
  */
  public boolean mConstraintEnabled, mRegulationEnabled;

  /* this variable is set by the federate ambassador when a time advance
     is received; it should be cleared by the main loop as appropriate
  */
  public boolean mTimeAdvanceGrant;

  /* this variable is set to the last time that has been granted to the
     federate ambassador (including time-advances due to enabling regulation
     or constraint)
  */
  public double mCurrentTime;

  /* we're not really using this information, but let's define this callback
     and throw an exception just to exercise throwing an exception from
     Java back into C++
  */

  public RTIControlModule parentRModule;


  DebugHelper D = new DebugHelper();

  //-------------------------------------------------------------
```

```
//-----------------------------------------------------------------

public RTIControlModuleFedamb (RTIControlModule parent) {

  //This constructor is used mainly to set the handle
  //to the parent RTIControlModule

  D.dbgOut("Instanciated RTIControlModuleFedamb");

  parentRModule = parent;


}//end of contructor


//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void startRegistrationForObjectClass(int theClass)
  {

    D.dbgOut("RTIControlModuleFedamb.startRegistrationForObjectClass" );
    parentRModule.issueUpdates = true;

  }//end of startRegistrationForObjectClass

//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void stopRegistrationForObjectClass(int theClass)
  throws ObjectClassNotPublished {

  D.dbgOut("RTIControlModuleFedamb.stopRegistrationForObjectClass");
  parentRModule.issueUpdates = false;

}//end of stopRegistrationForObjectClass

//-----------------------------------------------------------------
//-----------------------------------------------------------------


public void timeConstrainedEnabled(byte[] newtime)
  throws FederateInternalError {
  mConstraintEnabled = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeConstrainedEnabled

//-----------------------------------------------------------------
//-----------------------------------------------------------------


public void timeRegulationEnabled(byte[] newtime)
  throws FederateInternalError {

  mRegulationEnabled = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeRegulationEnabled

//-----------------------------------------------------------------
//-----------------------------------------------------------------
```

```
public void timeAdvanceGrant(byte[] newtime) throws FederateInternalError {
  mTimeAdvanceGrant = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeAdvanceGrant

//----------------------------------------------------------------------
//----------------------------------------------------------------------

public void removeObjectInstance(int oid,
                                 byte[] time,
                                 String tag,
                                 EventRetractionHandle erh)
  throws ObjectNotKnown {

  /* we don't care about the time or event-retraction handle, so just
     invoke the two-argument variation
  */

  removeObjectInstance(oid, tag);

}//end of removeObjectInstance

//----------------------------------------------------------------------
//----------------------------------------------------------------------


public void removeObjectInstance(int oid,
                                 String tag)
  throws ObjectNotKnown {

  D.dbgOut("Method: RTIControlModuleFedamb.removeObjectInstance() "
           + oid);


  // remove the object ID from our hash table of remote instances

  CMIFExperimentParticipant tempInstance =
    (CMIFExperimentParticipant)parentRModule.
    remoteParticipantsTable.remove(new Integer(oid));

  /* if "remove" returns null, the object was not present, so throw
     an exception
  */

  if (tempInstance != null) {

    //remove the info and handle panel from the display

    parentRModule.parentCCenter.
      CCDisplay.
      removeParticipantControlPanel(tempInstance.getControlPanel());

  } else {

    throw new ObjectNotKnown("instance not present " + oid);

  }

}//end of removeObjectInstance
```

```
//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void receiveInteraction(int iclass,
                               ReceivedInteraction phvpset,
                               byte[] time,
                               String tag,
                               EventRetractionHandle erh)
  throws InteractionParameterNotKnown, InteractionClassNotKnown,
         FederateInternalError {

  /* we don't care about the time or event-retraction handle, so just
     invoke the three-argument variation
  */
  receiveInteraction(iclass, phvpset, tag);

}//end of receiveInteraction

//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void receiveInteraction(int iclass,
                               ReceivedInteraction phvpset,
                               String tag)
  throws InteractionParameterNotKnown, InteractionClassNotKnown,
         FederateInternalError {

  if (iclass == RTIControlModule.hMessageToBus) {

    parentRModule.receiveMessageToBus(phvpset,tag);
    return;

  } else if (iclass == RTIControlModule.hStatusMessage){

    parentRModule.receiveStatusMessage(phvpset,tag);

    return;

  } else {

    /* all we know about are "Communication" interactions, so throw an
       exception
    */

    throw new InteractionClassNotKnown("class id is " + iclass);

  }

}//end of receiveInteraction

//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void discoverObjectInstance(int oid,
                                   int oclass,
                                   String theObjectName)
  throws ObjectClassNotKnown {

  D.dbgOut("Method: RTIControlModuleFedamb.discoverObjectInstance"
           + oid +", "
           + theObjectName);
```

```
    if (oclass != parentRModule.hExperimentParticipant)
      throw new ObjectClassNotKnown("unknown object class handle");

    CMIFExperimentParticipant tmpInstance =
      new CMIFExperimentParticipant(oid, parentRModule);

    try {

      tmpInstance.initialize(parentRModule.rtiamb);
      tmpInstance.initGraphicInfo();

    } catch (Exception e) {

      D.dbgOut("Exception in discoverObjectInstance: " + e.getMessage());

    }
    parentRModule.
      remoteParticipantsTable.put(new Integer(oid),
                                  tmpInstance);

    parentRModule.parentCCenter.
      CCDisplay.
      addParticipantControlPanel(tmpInstance.getControlPanel());

  }//end of discoverObjectInstance

  //---------------------------------------------------------------
  //---------------------------------------------------------------

  public void reflectAttributeValues(int oid,
                                     ReflectedAttributes ahvpset,
                                     byte[] time,
                                     String theTag,
                                     EventRetractionHandle rth)
    throws ObjectNotKnown, AttributeNotKnown, FederateInternalError {

    /* we don't care about the time or event-retraction handle, so just
       hand everything off to the RTIControlModule...without the
       time or event retraction handle
    */

    D.dbgOut("Method: RTIControlModuleFedamb.reflectAttributeValues()");

    parentRModule.reflectAttributeValues(oid, ahvpset, theTag);

  }//end of reflectAttributeValues

  //---------------------------------------------------------------
  //---------------------------------------------------------------

  public void reflectAttributeValues(int oid,
                                     ReflectedAttributes ahvpset,
                                     String theTag) throws ObjectNotKnown,
                                     AttributeNotKnown, FederateInternalError {

    //...hand the callback off to the RTIControlModule

    parentRModule.reflectAttributeValues(oid, ahvpset, theTag);

  }//end of reflectAttributeValues

}//end of CMIFTest01Fedamb
```

D.2 The package CMIFExperimentParticipant

D.2.1 The Class CMIFExperimentParticipant.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                  : CMIF, Kai Harth          /
//packages hla.rti13.java1    : U.S. DoD                 /
//packages java, javax        : Sun Microsystems         /
//-------------------------------------------------------/

package CMIF_HLA_Environment.CMIFExperimentParticipant;


import CMIF_HLA_Environment.CMIFControlCenter.*;
import CMIF_HLA_Environment.util.*;

import hla.rti13.java1.*;

import java.awt.event.*;
import java.awt.Color;
import java.awt.Dimension;
import java.util.*;
import java.math.BigInteger;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;

//=========================================================
//=========================================================

public class CMIFExperimentParticipant

{

  //This class is the backbone of every implementation
  //of an actual experiment participant.
  //It contains all the funcionlality to initiate the fedex, contact the
  //ControlCenter and so on...

  //---------------------------------
  //static variables

  //handles

  static int FEDEX_DOWN=1;
  static int FEDEX_SETUP=2;
  static int FEDEX_RUNNING=3;
  static int FEDEX_SHUTDOWN=4;
  static int FEDEX_CORRUPT=5;

  static int SIM_DOWN=6;
```

```
static int SIM_READY=7;
static int SIM_RUNNING=8;
static int SIM_DONE=9;
static int SIM_CORRUPT=10;
static int SIM_ABORTED=11;

static int PARTICIPANT_BUSY=10;
static int PARTICIPANT_IDLE=11;
static int PARTICIPANT_CORRUPT=12;


static int hRTIControlModule, hExperimentState, hExperimentParticipant,
  hParticipantName, hParticipantState, hControlInteraction,
  hControlMessageContent, hControlMessageRecipient, hMessageToBus,
  hToBusMessageContent, hToBusSender, hToBusRecipient, hMessageFromBus,
  hFromBusMessageContent, hFromBusSender, hFromBusRecipient,
  hStatusMessage, hStatusMessageContent, hStatusMessageSender;

/* keep a reference to the RTIambassador for use from within TestNode
   methods */

protected static RTIambassador sRtiAmb;

//-------------------------------------
//instance variables:

DebugHelper D = new DebugHelper ();

RTIControlModule parentRModule;

public RTIambassador rtiamb;

CMIFExperimentParticipantFedamb fedamb;

public String participantName = "unnamed";

public int participantID;

public int controlModuleID;

public int participantState = PARTICIPANT_IDLE;

public int controlModuleState;

public boolean issueUpdates = false;

public boolean controlCenterPresent = false;

public boolean timingIsSet = false;
public boolean runTickThread = true;
public boolean shutdownInitiated = false;

public boolean advanceRequested = false;
public boolean advanceGranted = false;
public double newFedambTime = 0.0;

int simState = SIM_DOWN;

int fedexState = FEDEX_DOWN;

//timing section
```

```
public String startTimeString = new String("0.0");
public String endTimeString = new String("100.0");
public String intervalTimeString = new String("1.0");
public String scaleString = new String("1.0");

public double startTimeDouble = 0.0;
public double endTimeDouble = 100.0;
public double intervalTimeDouble = 1.0;
public double scaleDouble = 1.0;

//--------------------
//graphical components for the ddisplay section in the
//controlDisplay

JPanel participantControlPanel, infoPanel, buttonPanel;

JLabel nameLabel, stateIconLabel;

Border lineBorder = BorderFactory.createLineBorder(Color.black, 1);
Border emptyBorder = BorderFactory.createEmptyBorder(8,8,8,8);

//icons and bullets

String path = new String("CMIF_HLA_Environment/CMIFControlCenter/icons/");

ImageIcon greenIcon = new ImageIcon(path+"grn.gif");
ImageIcon greenIconPulse = new ImageIcon(path+"grn_pulse.gif");
ImageIcon redIcon = new ImageIcon(path+"red.gif");
ImageIcon redIconPulse = new ImageIcon(path+"red_pulse.gif");
ImageIcon orangeIcon = new ImageIcon(path+"orange.gif");
ImageIcon orangeIconPulse = new ImageIcon(path+"orange_pulse.gif");

JButton shutDownButton, resetButton, stopButton, pingButton;


//=========================================================
//=========================================================
//local section, to use for independent instances

public CMIFExperimentParticipant(String name)

  {
    //This contructor is used to instanciate CMIFExperimentParticipant
    //as a superclass for an independent implementation of an actual
    //simualtion participant

    D.dbgOut("Started ExperimentParticipant..... " + name);
    D.setDebugState(DebugHelper.FULL_DEBUG);

    participantName = name;

  }//end of independent constructor

//=========================================================
//=========================================================

public void doSetup() {

  //This method uses an own thread to run the methods
  //launchRTI() and runPreSimulation()

  final SwingWorker worker = new SwingWorker() {
```

```
    public Object construct() {

      //...code that might take a while
      //to execute is here...

      launchRTI();
      runPreSimulation();

      D.dbgOut("$#" + participantName +
               " is done with doSetup Thread");
      return null;

    }

  };

  worker.start();

  runBackgroundTickThread();

}//end of launchRTIInOwnThread

//=============================================================
//=============================================================

public void launchRTI() {

  //This method kicks off the RTI execution

  fedexState=FEDEX_SETUP;

  D.dbgOut("$Method: CMIFExperimentParticipant.launchRTI() ");

  fedamb = new CMIFExperimentParticipantFedamb(this);

  try {

    // construct the RTI ambassador
    rtiamb = new RTIambassador();

    /* The "fedex" sets itself up asynchronously with respect to the
       return of the "createFederationExecution" call, so join attempts
       occuring quickly afterwards will fail. If this happens, we'll keep
       looping and trying.
    */

    int tries = 30;
    while (tries > 0) {
      try {
        rtiamb.joinFederationExecution(participantName,
                                       "CMIF_HLA_Environment",
                                       fedamb);
        break;
      } catch (FederationExecutionDoesNotExist ex) {
        D.dbgOut(participantName + ": federation does not exist");

        // decrement the number of remaining tries and
        //tick to kill some time

        tries--;
        rtiamb.tick(0.4, 0.5);
      }
    }
```

```
    /* If we still haven't joined, try once more and let the exception
       propagate. */

    if (tries == 0)
      rtiamb.joinFederationExecution(participantName,
                                     "CMIF_HLA_Environment",
                                     fedamb);

    D.dbgOut(participantName + ": joined federation execution");


  }
  catch (RTIexception ex) {

    /* catch exceptions that result from creating the RTI ambassador,
       creating the federation executive, or joining the federation
       execution */

    D.dbgOut(participantName + ": caught exception " + ex);

    fedexState=FEDEX_CORRUPT;

    return;
  }

  try {

    initialize(rtiamb);
    D.dbgOut(participantName +
             ": established RTI<->SIM handle mapping");

    publishAndSubscribe();
    D.dbgOut(participantName + ": initialized DM disposition");

    participantID =
      rtiamb.registerObjectInstance(hExperimentParticipant);

    D.dbgOut(participantName + ": registered Object Instance, ID: " +
             participantID);

    // enable time-constraint

    rtiamb.enableTimeConstrained();

    /* our federate ambassador will set a boolean variable to "true"
       when it receives notification that constraint has been enabled;
       we'll loop and tick until this happens
    */

    while (!fedamb.mConstraintEnabled) {
      rtiamb.tick(0.01, 0.2);
    }

    D.dbgOut(participantName + ": time constraint enabled @ " +
             fedamb.mCurrentTime);

    rtiamb.enableAsynchronousDelivery();
    D.dbgOut(participantName + ": asynchronous delivery enabled");

    rtiamb.tick();
```

```
    } catch (Exception e) {

      D.dbgOut("Exception occured in launchRTI: " +
               e.getMessage());

    }

  }//end of launchRTI

  //============================================================
  //============================================================

  public void runPreSimulation() {

    D.dbgOut("$Method: CMIFExperimentParticipant.runPreSimulation() ");

    try {

      //wait until the control center is located and listening
      //for updates...then send the first update

      while (!issueUpdates) {
        tick(0.1, 0.2);
      }

      updateAttributeValues( getCurrentTime() );

      tick();

      //wait for the first update from the control center

      while (!controlCenterPresent)
        tick();

      fedexState=FEDEX_RUNNING;
      D.dbgOut("$#"+ participantName +" is in contact with the "+
               "Control Center");

      //now wait and tick until timing parameters have arrived,
      //then call timingParametersChanged and doneWithSetup() to
      //let the implementation know that it can proceed

      while (!timingIsSet)
        tick();

      timingParametersChanged();

      simState = SIM_READY;

      updateAttributeValues(fedamb.mCurrentTime);
      doneWithSetup();

      D.dbgOut("$#"+ participantName +" is done with runPreSimulation");

    } catch (Exception e) {

      fedexState = FEDEX_CORRUPT;

      D.dbgOut("Exception occured in launchRTI: " +
               e.getMessage());
```

```
  }

}//end of runPreSimulation

//=======================================================
//=======================================================

public void advanceOneStep() {

  //this method is to be called if the advancement in time
  //should follow the given increments.

  double newTime = fedamb.mCurrentTime;

  int iter=1;

  while (newTime>=(startTimeDouble+iter*intervalTimeDouble)){
    iter++;
  }

  newTime=startTimeDouble+iter*intervalTimeDouble;

  D.dbgOut("$#fedambtime is: " + fedamb.mCurrentTime + ", trying to go to "
           + newTime);

  advanceTo(newTime);

}//end of advanceOneStep

//=======================================================
//=======================================================

public void advanceTo( double newTime) {

  //..sets all values and flags so that the
  //simulation thread will try to advance to the newTime.
  //Feasibility of the newTime value is also checked

  if (newTime>fedamb.mCurrentTime && newTime<=endTimeDouble) {

    newFedambTime = newTime;

    advanceGranted=false;
    advanceRequested=true;

    D.dbgOut(participantName +" waiting for new timing value ");

    while (!advanceGranted);

    updateAttributeValues(fedamb.mCurrentTime);

  } else if (newTime>=endTimeDouble) {

    D.dbgOut(participantName +" has reached simulation end time!");
    simState=SIM_DONE;
    updateAttributeValues(fedamb.mCurrentTime);
    doneWithSimulation();

  } else {

    D.dbgOut(participantName +" requested non feasible time value");
```

```
  }
}//end of advanceTo

//=====================================================
//=====================================================

public boolean simIsRunning() {

  return (simState==SIM_RUNNING);

}//end of simIsRunning

//=====================================================
//=====================================================

public void shutDownRTI() {

  //This method tries to resign from the federation execution

  D.dbgOut("Method:CMIFExperimentParticipant.shutDownRTI");

  if ((fedexState==FEDEX_SHUTDOWN) && simState!=SIM_RUNNING) {

    try {

      //try to kill the instance

      rtiamb.deleteObjectInstance(participantID,
                                  EncodingHelpers.encodeDouble
                                  (fedamb.mCurrentTime), "");

      rtiamb.tick();

      // try to resign

      rtiamb.resignFederationExecution
        (ResignAction.DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES);
      D.dbgOut(participantName + ": resigned from federation");

      fedexState=FEDEX_DOWN;

      doneWithShutDown();

    }

    catch (RTIexception ex) {

      D.dbgOut(participantName +
               ": caught exception while resigning " + ex);
      fedexState=FEDEX_CORRUPT;
      return;

    }
  }

}//end of shutDownRTI

//=====================================================
//=====================================================

public static void initialize(RTIambassador rtiamb) throws RTIexception {
```

```
    // hold onto a reference to our RTI ambassador
    sRtiAmb = rtiamb;

    // obtain RTI handles to the FED entities used in the federation

    hRTIControlModule = rtiamb.getObjectClassHandle("RTIControlModule");
    hExperimentState =  rtiamb.getAttributeHandle("ExperimentState",
                                                  hRTIControlModule);

    hExperimentParticipant =
      rtiamb.getObjectClassHandle("CMIFExperimentParticipant");
    hParticipantName =  rtiamb.getAttributeHandle("ParticipantName",
                                                  hExperimentParticipant);
    hParticipantState =  rtiamb.getAttributeHandle("ParticipantState",
                                                   hExperimentParticipant);


    hControlInteraction =
      rtiamb.getInteractionClassHandle("ControlInteraction");
    hControlMessageContent =
      rtiamb.getParameterHandle("ControlMessageContent",
                                hControlInteraction);
    hControlMessageRecipient =
      rtiamb.getParameterHandle("ControlMessageRecipient",
                                hControlInteraction);

    hMessageToBus= rtiamb.getInteractionClassHandle("MessageToBus");
    hToBusMessageContent=
      rtiamb.getParameterHandle("ToBusMessageContent",hMessageToBus );
    hToBusSender=
      rtiamb.getParameterHandle("ToBusSender",hMessageToBus );
    hToBusRecipient=
      rtiamb.getParameterHandle("ToBusRecipient",hMessageToBus );


    hMessageFromBus= rtiamb.getInteractionClassHandle("MessageFromBus");
    hFromBusMessageContent=
      rtiamb.getParameterHandle("FromBusMessageContent",hMessageFromBus );
    hFromBusSender=
      rtiamb.getParameterHandle("FromBusSender",hMessageFromBus );
    hFromBusRecipient=
      rtiamb.getParameterHandle("FromBusRecipient",hMessageFromBus );


    hStatusMessage= rtiamb.getInteractionClassHandle("StatusMessage");
    hStatusMessageContent=
      rtiamb.getParameterHandle("StatusMessageContent",hStatusMessage );
    hStatusMessageSender=
      rtiamb.getParameterHandle("StatusMessageSender",hStatusMessage );

  }//end of initialize

  //----------------------------------------------------------------
  //----------------------------------------------------------------

  public static void publishAndSubscribe() throws RTIexception {


    //PUBLISHING
    //create an outgiong attribute-handle set
```

```
    AttributeHandleSet ahsetOut = AttributeHandleSetFactory.create(2);

    // populate the set with the attributes we publish/subscribe

    ahsetOut.add(hParticipantName);
    ahsetOut.add(hParticipantState);

    // perform the appropriate DM calls
    sRtiAmb.publishObjectClass(hExperimentParticipant, ahsetOut);

    sRtiAmb.publishInteractionClass(hStatusMessage);
    sRtiAmb.publishInteractionClass(hMessageToBus);

    //SUBSCRIBING
    //create an attribute-handle set for experiment participant

    AttributeHandleSet ahsetIn = AttributeHandleSetFactory.create(1);

    ahsetIn.add(hExperimentState);

    // perform the appropriate DM calls

    sRtiAmb.subscribeObjectClassAttributes(hRTIControlModule, ahsetIn);

    sRtiAmb.subscribeInteractionClass(hMessageFromBus);
    sRtiAmb.subscribeInteractionClass(hControlInteraction);

  }//end of publishAndSubscribe

  //=========================================================
  //=========================================================

  public void updateAttributeValues (double thetime) {

    //This method is called to send changes attributes on the way.
    //These are published via the RTI and reach all recipients, where in
    //the end reflectAttributeValues() of any local representations is
    //called to adopt the changes

    try {

      D.dbgOut("Method: CMIFExperimentParticipant.updateAttributeValues()");
      D.dbgOut("# fedexState "+fedexState+", simState: "+simState);

      // create a set for the outgoing attributes

      SuppliedAttributes ahvpset = SuppliedAttributesFactory.create(2);

      ahvpset.add(hParticipantName,
                  EncodingHelpers.encodeString(participantName));
      ahvpset.add(hParticipantState,
                  EncodingHelpers.encodeInt(10*fedexState +simState ));

      // send the update on its way
      sRtiAmb.updateAttributeValues(participantID,
                                    ahvpset,
                                    "Java" );
    } catch (Exception e) {

      D.dbgOut("Exception in updateAttributeValues :" + e.getMessage());

    }
```

```
}//end of updateAttributeValues


//=========================================================
//=========================================================

public void receiveControlInteraction(ReceivedInteraction phvpset,
                                       String tag) {

  //This method is called when the Federate Ambassador receives
  //any control interaction.
  //Then it has to be evaluated, if the message is intended for this
  //specific participant and if, the content has to be processed.

  D.dbgOut(participantName + " received ControlInteraction, " + tag);

  try {

    String controlMessageRecipient =
      EncodingHelpers.decodeString(phvpset.getValue(1));
    D.dbgOut("$#....for: " + controlMessageRecipient);

    if (controlMessageRecipient.equals(participantName) ||
        controlMessageRecipient.equals("all")) {

      String controlMessage =
        EncodingHelpers.decodeString(phvpset.getValue(0));

      D.dbgOut("$#....message: " + controlMessage);

      //--------------------------

      if (controlMessage.equals("shutdown")){
        fedexState=FEDEX_SHUTDOWN;
      }

      //--------------------------

      if (controlMessage.equals("ping")){

        sendStatusMessage(reactToPing());

      }

      //--------------------------
      if (controlMessage.startsWith("timing")){

        evaluateTimingMessage(controlMessage);

      }

      //--------------------------
      if (controlMessage.equals("start_simulation")){

        simState = SIM_RUNNING;

      }

      //--------------------------
      if (controlMessage.equals("abort_simulation")){
```

```
        simState =SIM_ABORTED;
        abortSimulation();

      }
    }


  } catch (Exception e){

    D.dbgOut("Exception in CMIFExperimentParticipant."+
             "receiveControlInteraction:" );
    D.dbgOut("#" + e.toString());

  }


}//end of receiveControlInteraction

//=============================================================
//=============================================================

public void receiveMessageFromBus(ReceivedInteraction phvpset,
                                  String tag) {

  //This method is called when the Federate Ambassador receives
  //any message from a bus.
  //Then it has to be evaluated, if the message is intended for this
  //specific participant and if, the content has to be processed.

  D.dbgOut(participantName + " received , MessageFromBus" + tag);

  try {

    String messageRecipient =
      EncodingHelpers.decodeString(phvpset.getValue(1));

    if(messageRecipient.equals(participantName)||
       messageRecipient.equals("all")) {

      String messageSender =
        EncodingHelpers.decodeString(phvpset.getValue(2));

      String message =
        EncodingHelpers.decodeString(phvpset.getValue(0));

      D.dbgOut("$#...sender: " +messageSender +
               ", recipient: " +messageRecipient+
               " ,message: " + message);

      receivedMessageOnBus(messageSender, message);

    }

  } catch (Exception e){

    D.dbgOut("Exception in CMIFExperimentParticipant."+
             "receiveMessageFromBus:" );
    D.dbgOut("#" + e.toString());

  }
```

```
}//end of receiveMessageFromBus

//----------------------------------------------------------------
//----------------------------------------------------------------

public void sendStatusMessage(String message)  {

  //This method is used to send a status message
  //represented by the string 'message' to the control center.

  try {

    D.dbgOut(participantName +" is Sending StatusMessage");

    // create the outgoing parameters set
    SuppliedParameters phvpset = SuppliedParametersFactory.create(2);


    // add the bytes to the parameter set
    phvpset.add(hStatusMessageSender,
                EncodingHelpers.encodeString(participantName));
    phvpset.add(hStatusMessageContent,
                EncodingHelpers.encodeString(message));

    // send the interaction on its way
    sRtiAmb.sendInteraction(hStatusMessage, phvpset, participantName);

  } catch (Exception e) {

    D.dbgOut(participantName + " Exception in "+
             "sendStatusMessage:" );
    D.dbgOut("#" + e.toString());

  }

}//end of sendStatusMessage

//============================================================
//============================================================
//----------------------------------------------------------------
//----------------------------------------------------------------

public void sendMessageToBus(String message, String recipient)  {

  //This method is used to send a  message
  //represented by the string 'message' to the communication bua.

  try {

    D.dbgOut(participantName +" is Sending MessageToBus");

    // create the outgoing parameters set
    SuppliedParameters phvpset = SuppliedParametersFactory.create(3);


    // add the bytes to the parameter set
    phvpset.add(hToBusSender,
                EncodingHelpers.encodeString(participantName));
    phvpset.add(hToBusRecipient,
                EncodingHelpers.encodeString(recipient));
```

```
    phvpset.add(hToBusMessageContent,
                EncodingHelpers.encodeString(message));

    // send the interaction on its way
    sRtiAmb.sendInteraction(hMessageToBus, phvpset, participantName);

  } catch (Exception e) {

    D.dbgOut(participantName + " Exception in "+
             "sendMessageToBus:" );
    D.dbgOut("#" + e.toString());

  }

}//end of sendMessageToBus

//=============================================================
//=============================================================

public double getCurrentTime() {

  //This method can be used from a subclass of ExperimentParticipant.
  //This way the user does not have to know about and access the
  //Fedeate Ambassador

  return fedamb.mCurrentTime;

}//end of getCurrentTime

//=============================================================
//=============================================================

public void tick(double time1 , double time2 ) {

  //This method invokes the RtiAmbassador mehtod of the
  //same name. So the the user implementing subclasses of
  //ExperimentParticipant does not have to know about the
  //RtiAmb mechanisms and exceptions.
  //tick() in general grants the rtiamb time to do operations.

  try {

    rtiamb.tick(time1, time2);

  } catch (Exception e) {

    D.dbgOut("Exception while trying to tick the rtiamb :" +
             e.getMessage() );

  }

}//end of tick

//=============================================================
//=============================================================

public void tick() {

  //...as above, just without the max and min times.

  try {
```

```
    rtiamb.tick();

  } catch (Exception e) {

    D.dbgOut("Exception while trying to tick the rtiamb :" +
             e.getMessage() );

  }

}//end of tick


//-------------------------------------------
//-------------------------------------------

private void evaluateTimingMessage(String message) {

  //This method receives as input the encoded timing string
  //sent by the control center

  String delim = "#";

  StringTokenizer messageTokenizer =
    new StringTokenizer(message, delim);

  messageTokenizer.nextToken();

  while(messageTokenizer.hasMoreTokens() ) {

    convertTimeStrings(messageTokenizer.nextToken(),
                       messageTokenizer.nextToken() );

  }

  timingIsSet = true;
  timingParametersChanged();

}//end of evaluateTimingMessage

//-------------------------------------------
//-------------------------------------------

private void convertTimeStrings(String whichOne, String stringTime) {

  //This method attempts to convert the string inputs made for time management
  //(either from file or from the text boxes) to the neccessare double values.
  //It returns (true/false) if the attempt was successful...

  double tmpDouble=0.0;

  tmpDouble = (new Double(stringTime)).doubleValue();

  //D.dbgOut("$#converted to: " + tmpDouble);

  if (whichOne.equals("start")){
    startTimeDouble = tmpDouble;
    startTimeString= (new Double(startTimeDouble)).toString();
  }

  if (whichOne.equals("end")){
    endTimeDouble = tmpDouble;
    endTimeString= (new Double(endTimeDouble)).toString();
```

```
    }

    if (whichOne.equals("interval")) {
      intervalTimeDouble = tmpDouble;
      intervalTimeString= (new Double(intervalTimeDouble)).toString();
    }

    if (whichOne.equals("scale") ){
      scaleDouble = tmpDouble;
      scaleString= (new Double(scaleDouble)).toString();
    }

  }//end of convertTimeStrings

  //=======================================================
  //=======================================================

  private void runBackgroundTickThread() {

    //This method is started in the beginning and ensures, that whenever
    //no other phase is using (and thus ticking) the RTI, ticks are sent.

    final SwingWorker worker = new SwingWorker() {
      public Object construct() {

        //background code goes here

        D.dbgOut("Starting background tick thread!");

        while (runTickThread) {

          if (fedexState==FEDEX_SHUTDOWN && !shutdownInitiated)
            {
              shutdownInitiated = true;
              shutDownRTI();
            }

          if (fedexState==FEDEX_RUNNING) {

            //if ((simState==SIM_READY)||(simState==SIM_ABORTED)||
            //(simState==SIM_CORRUPT)) {

            if (simState!=SIM_RUNNING && simState!=SIM_DOWN) {

              try {

                rtiamb.tick(0.1, 0.3);

              } catch (Exception e) {

                D.dbgOut(participantName +
                         "Exception in background tick thread:");
                D.dbgOut("...:" + e.getMessage());

              }
            }
          }


          //--------------------------------
          //--------------------------------
          //--------------------------------
```

```
//-------------------------------
//-------------------------------
//
//D U R I G    S I M U L A T I O N .....

if (simState==SIM_RUNNING) {

  D.dbgOut("$# " +participantName +
           ": entered simulation background thread");

  try {

    //-----------------------------------------------
    //set fedamb time to desired simulation start time

    fedamb.mTimeAdvanceGrant=false;

    rtiamb.timeAdvanceRequest( EncodingHelpers.encodeDouble
                               (startTimeDouble) );
    D.dbgOut("$# requested timeAdvanceRequest");

    // loop and tick until we receive the time advance

    while (!fedamb.mTimeAdvanceGrant) {
      rtiamb.tick(0.01, 0.3);
    }

    D.dbgOut("$# " +participantName +": Set fedamb to start time");

    updateSimTime(fedamb.mCurrentTime);

  } catch (Exception e) {

    D.dbgOut(participantName +" Exception in runSimulation()");
    D.dbgOut("# " + e.toString());

    simState=SIM_CORRUPT;

  }


  try {

    while (simState==SIM_RUNNING) {

      if (advanceRequested) {

        D.dbgOut("$#" +participantName +
                 " trying to set fedamb to time " +newFedambTime);

        fedamb.mTimeAdvanceGrant=false;

        rtiamb.timeAdvanceRequest( EncodingHelpers.encodeDouble
                                   (newFedambTime ));

        // loop and tick until we receive the time advance

        while (!fedamb.mTimeAdvanceGrant) {
          rtiamb.tick(0.01, 0.3);
        }
```

```
                advanceGranted = true;
                advanceRequested = false;

                D.dbgOut("$#" +participantName +" Set fedamb to time " +
                         fedamb.mCurrentTime);

                updateSimTime(fedamb.mCurrentTime);

              }

              rtiamb.tick();

            }//end of while

          } catch (Exception e) {

            D.dbgOut(participantName +" Exception in Simulation thread!");
            D.dbgOut("# " + e.toString());

            simState=SIM_CORRUPT;

          }

        }

        //-----------------------------------
        //-----------------------------------
        //-----------------------------------

      }//end of while

      D.dbgOut(participantName +
               "Stopped background tick thread!");
      return null;
    }
  };
  worker.start();

}//end of runBackgroundTickThread

//============================================================
//============================================================

//****************************** * * * *  *  *  *    *   *

//============================================================
//============================================================


//remote section (used by the RTIContolModule to keep reference
//with remote instances)

public CMIFExperimentParticipant(int oid, RTIControlModule parent) {

  //This constructor is used from the RTIControlModule to instanciate
  //representations of the remote experiment participants

  D.dbgOut("Started ExperimentParticipant..... " + oid);
  D.setDebugState(DebugHelper.FULL_DEBUG);

  participantID = oid;
  parentRModule = parent;
```

```
}//end of remote constructor

//=========================================================
//=========================================================

public void reflectAttributeValues(ReflectedAttributes ahvpset) {

  //This method updates the attribute values of an instance kept at the
  //controlModule. It is at the "receiving end" of the update process
  //initiated by the updateAttributeValues() method above, which is used
  //in the independent instances of an Experimentparticipant


  //now, analyze the attribute handle set

  D.dbgOut("Method:CMIFExperimentParticipant.reflectAttributeValues()");

  try {

    int size = ahvpset.size();

    // loop through the handle-value pairs in the set

    for (int a = 0; a < size; a++) {

      // obtain the value of the handle

      int attributeHandle = ahvpset.getHandle(a);

      if (attributeHandle == hParticipantName) {
        D.dbgOut("$#reflectAttributeValues: found participantName");
        participantName =
          EncodingHelpers.decodeString(ahvpset.getValue(a));
      }

      else if (attributeHandle == hParticipantState) {
        participantState =
          EncodingHelpers.decodeInt(ahvpset.getValue(a));

        D.dbgOut("$#reflectAttributeValues: found participantState" +
                 participantState);

        BigInteger stateBI = BigInteger.valueOf(participantState);

        fedexState= (stateBI.divide(BigInteger.valueOf(10))).intValue();
        simState= (stateBI.mod(BigInteger.valueOf(10))).intValue();
        D.dbgOut("$#simState: " +simState+", fedexState: " +fedexState);

      }

      else

        // all we know is name and state, so throw an exception
        throw new AttributeNotKnown("attribute handle is " +
                                    attributeHandle);
    }

    updateControlPanel();
```

```
    } catch (Exception e) {
      D.dbgOut("Exception in reflectAttributeValues :" + e.getMessage());
    }

  }//end of reflectAttributeValues

  //============================================================
  //============================================================

  public void initGraphicInfo() {

    //This method puts together the information panel
    //that allows access to each located participant.
    //It will be incorporated into the participants pane
    //in the control Display...

    D.dbgOut("Method:CMIFExperimentParticipant.initGraphicInfo()");


    //----------------
    //info section

    nameLabel = new JLabel(participantName);
    nameLabel.setForeground(Color.black);

    stateIconLabel = new JLabel(orangeIcon);

    infoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

    infoPanel.add(stateIconLabel);

    infoPanel.add(new JLabel("Name: "));
    infoPanel.add(nameLabel);

    //----------------
    //button section

    buttonPanel = new JPanel();

    shutDownButton = new JButton("Shut Down");
    shutDownButton.setToolTipText("...forces to resign from federation");
    shutDownButton.addActionListener((ActionListener)parentRModule);

    resetButton = new JButton("Reset");
    resetButton.setToolTipText("...resets to inital state");
    stopButton = new JButton("Stop");
    stopButton.setToolTipText("...stops current activity");
    pingButton = new JButton("Ping");
    pingButton.setToolTipText("...pings for status report");
    pingButton.addActionListener((ActionListener)parentRModule);

    buttonPanel.add(shutDownButton);
    buttonPanel.add(pingButton);
    buttonPanel.add(stopButton);
    buttonPanel.add(resetButton);

    //----------------
    //together
```

```
    participantControlPanel = new JPanel();

    participantControlPanel.
      setBorder(new CompoundBorder(emptyBorder,lineBorder));

    participantControlPanel.add(infoPanel);
    participantControlPanel.add(Box.createHorizontalGlue());
    participantControlPanel.add(buttonPanel);

    participantControlPanel.setMaximumSize(new Dimension(600,160));

  }//end of initGraphicInfo

  //============================================================
  //============================================================

  private void updateControlPanel(){

    //This method updates the information displayed
    //in the control panel to reflect the attribute values
    //of this participant

    nameLabel.setText(participantName);
    shutDownButton.setActionCommand("shutdown@" + participantName);
    pingButton.setActionCommand("ping@" + participantName);

    updateStateIconLabel();

    participantControlPanel.revalidate();

  }//end of updateControlPanel

  //============================================================
  //============================================================

  public void updateStateIconLabel(){

    if (simState==SIM_DOWN)
      stateIconLabel.setIcon(orangeIcon);
    if (simState==SIM_READY)
      stateIconLabel.setIcon(greenIcon);
    if (simState==SIM_RUNNING)
      stateIconLabel.setIcon(greenIconPulse);
    if (simState==SIM_CORRUPT || fedexState==FEDEX_CORRUPT)
      stateIconLabel.setIcon(redIconPulse);
    if (simState==SIM_ABORTED )
      stateIconLabel.setIcon(redIconPulse);
    if (simState==SIM_DONE )
      stateIconLabel.setIcon(redIcon);

    stateIconLabel.revalidate();

  }//end of updateStateIconLabel

  //============================================================
  //============================================================

  public JPanel getControlPanel() {

    //...convenience method to get a handle to the
    //handle Panel of this participant.
```

```
  D.dbgOut("Method:CMIFExperimentParticipant.getControlPanel()");

  return participantControlPanel;

}//end of getControlPanel

//=============================================================
//=============================================================
//+ + + + +  + + + + + +  + + + + + +  + + + + +  + + +
//=============================================================
//=============================================================

//The following methods are used for callbacks and are empty
//They have to be overridden in the participant implementation


//=============================================================
//=============================================================

public void receivedMessageOnBus( String sender,
                                  String content) {

  //Callback after a message was received

}//end of receivedMessageOnBus

//=============================================================
//=============================================================

public void doneWithSetup() {

  //This method is called after the federate is all set up,
  //in contact with the ControlCenter and supplied with neccessary
  //information for the simulation run

}//end of doneWithSetup

//=============================================================
//=============================================================

public void doneWithShutDown(){


  //This is the callback after the federation is successfully
  //resigned

}//end of  doneWithShutDown

//=============================================================
//=============================================================

public void doneWithSimulation(){

  //This is the callback after the federation is successfully
  //resigned

}//end of  doneWithSimulation

//=============================================================
//=============================================================

public String reactToPing() {
```

```
    //A callback after receiving a ping request from the ControlCenter
    //The string that has to be returned will be sent to the ControlCenter
    //as a status message. Other activities can be implemented in
    //overriding this in the participant implementation

    String pingReactionString = new String(participantName +
                                           " is reacting to PING");

    return pingReactionString;

  }//end of reactToPing

  //=========================================================
  //=========================================================

  public void timingParametersChanged() {

    //This is a callback after the ControlCenter has sent
    //timing information (start, stop, scale and increment).
    //The internal processing is already done. This is to notify
    //the implementation of the new situation.

  }//end of timingParametersChanged

  //=========================================================
  //=========================================================

  public void updateSimTime(double simTime) {

    //This callback can be overriden to do something each time a new
    ///simulation time is set...

  }//end of updateSimTime


  //=========================================================
  //=========================================================

  public void waitForSimulation() {

    while (simState!=SIM_RUNNING);

  }//end of waitForSimulation

  //=========================================================
  //=========================================================

  public void abortSimulation() {

    //This is a callback after the ControlCenter has sent
    //the signal to abort the simulation

  }//end of abortSimulation

}//end of class CMIFExperimentParticipant
```

D.2.2 The Class CMIFExperimentParticipantFedamb.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                  : CMIF, Kai Harth          /
//packages hla.rti13.java1    : U.S. DoD                 /
//packages java, javax        : Sun Microsystems         /
//-------------------------------------------------------/

package CMIF_HLA_Environment.CMIFExperimentParticipant;

import CMIF_HLA_Environment.CMIFControlCenter.*;
import CMIF_HLA_Environment.util.*;

// import the RTI classes into our namespace
import hla.rti13.java1.*;

/* we only implement a small subset of FederateAmbassador methods, so derive
   our ambassador from NullFederateAmbassador to pick up no-op implementations
   of the others
*/

class CMIFExperimentParticipantFedamb extends NullFederateAmbassador {


  /* these variables are checked by the main loop to detect when regulation
     and constraint have been enabled
  */
  public boolean mConstraintEnabled, mRegulationEnabled;

  /* this variable is set by the federate ambassador when a time advance
     is received; it should be cleared by the main loop as appropriate
  */

  public boolean mTimeAdvanceGrant;

  /* this variable is set to the last time that has been granted to the
     federate ambassador (including time-advances due to enabling regulation
     or constraint)
  */

  public double mCurrentTime;

  /* we're not really using this information, but let's define this callback
     and throw an exception just to exercise throwing an exception from
     Java back into C++
  */

  public CMIFExperimentParticipant parentParticipant;

  DebugHelper D = new DebugHelper();

  //-----------------------------------------------------------------
  //-----------------------------------------------------------------
```

```
public CMIFExperimentParticipantFedamb (CMIFExperimentParticipant parent) {

  //This constructor is used mainly to set the handle
  //to the parent RTIControlModule

  D.dbgOut("Instanciated CMIFExperimentParticipantFedamb");

  parentParticipant = parent;


}//end of contructor


//----------------------------------------------------------------
//----------------------------------------------------------------


public void startRegistrationForObjectClass(int theClass)
  {

    D.dbgOut("CMIFExperimentManagerFedamb.startRegistrationForObjectClass");
    parentParticipant.issueUpdates = true;

  }//end of startRegistrationForObjectClass

//----------------------------------------------------------------
//----------------------------------------------------------------

public void stopRegistrationForObjectClass(int theClass)
  {

    D.dbgOut("CMIFExperimentManagerFedamb.stopRegistrationForObjectClass");
    parentParticipant.issueUpdates = false;

  }//end of startRegistrationForObjectClass


//----------------------------------------------------------------
//----------------------------------------------------------------

public void timeRegulationEnabled(byte[] newtime)
  throws FederateInternalError {

  mRegulationEnabled = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeRegulationEnabled

//----------------------------------------------------------------
//----------------------------------------------------------------

public void timeConstrainedEnabled(byte[] newtime)
  throws FederateInternalError {
  mConstraintEnabled = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeConstrainedEnabled

//----------------------------------------------------------------
//----------------------------------------------------------------
```

```
public void timeAdvanceGrant(byte[] newtime) throws FederateInternalError {

  mTimeAdvanceGrant = true;
  mCurrentTime = EncodingHelpers.decodeDouble( newtime );

}//end of timeAdvanceGrant

//----------------------------------------------------------------
//----------------------------------------------------------------

public void removeObjectInstance(int oid,
                                 byte[] time,
                                 String tag,
                                 EventRetractionHandle erh)
  throws ObjectNotKnown {

  /* we don't care about the time or event-retraction handle, so just
     invoke the two-argument variation
  */
  removeObjectInstance(oid, tag);

}//end of removeObjectInstance

//----------------------------------------------------------------
//----------------------------------------------------------------

public void removeObjectInstance(int oid,
                                 String tag)
  throws ObjectNotKnown {

  //This method reacts to a callback that the RTIControlModule is
  //no longer present....

  D.dbgOut("Method: RTIControlModuleFedamb.removeObjectInstance()");

  if (oid==parentParticipant.controlModuleID) {

    parentParticipant.controlModuleID=-1;
    parentParticipant.controlCenterPresent = false;

  } else {

    throw new ObjectNotKnown("unknown object class handle");

  }


}//end of removeObjectInstance

//----------------------------------------------------------------
//----------------------------------------------------------------

public void discoverObjectInstance(int oid,
                                   int oclass,
                                   String theObjectName)
  throws ObjectClassNotKnown {

  //This method is called when the federate discovers the
  //presence of the RTIControlModule.
  //Then, the object ID of the controlModule is noted for future
  //reference
```

```
  D.dbgOut("Method: CMIFExperimentParticipantFedamb.discoverObjectInstance");

  if (oclass != parentParticipant.hRTIControlModule)
    throw new ObjectClassNotKnown("unknown object class handle");

  parentParticipant.controlModuleID = oid ;
  parentParticipant.controlCenterPresent = true;

}//end of discoverObjectInstance

//----------------------------------------------------------------
//----------------------------------------------------------------

public void receiveInteraction(int iclass,
                               ReceivedInteraction phvpset,
                               byte[] time,
                               String tag,
                               EventRetractionHandle erh)
  throws InteractionParameterNotKnown, InteractionClassNotKnown,
         FederateInternalError {

  /* we don't care about the time or event-retraction handle, so just
     invoke the three-argument variation
  */
  receiveInteraction(iclass, phvpset, tag);

}//end of receiveInteraction

//----------------------------------------------------------------
//----------------------------------------------------------------

public void receiveInteraction(int iclass,
                               ReceivedInteraction phvpset,
                               String tag)

  throws InteractionParameterNotKnown, InteractionClassNotKnown,
         FederateInternalError {

  if (iclass == CMIFExperimentParticipant.hMessageFromBus) {

    parentParticipant.receiveMessageFromBus(phvpset,tag);

    return;

  } else if (iclass == CMIFExperimentParticipant.hControlInteraction){

    parentParticipant.receiveControlInteraction(phvpset,tag);

    return;

  } else {

    /* all we know about are "Communication" interactions, so throw an
       exception
    */

    throw new InteractionClassNotKnown("class id is " + iclass);

  }

}//end of receiveInteraction
```

```
//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void reflectAttributeValues(int oid,
                                   ReflectedAttributes ahvpset,
                                   byte[] time,
                                   String theTag,
                                   EventRetractionHandle rth)

  throws ObjectNotKnown, AttributeNotKnown, FederateInternalError {

  /* we don't care about the time or event-retraction handle, so just
     invoke the three-argument variation
  */

  reflectAttributeValues(oid, ahvpset, theTag);

}//end of reflectAttributeValues

//-----------------------------------------------------------------
//-----------------------------------------------------------------

public void reflectAttributeValues(int oid,
                                   ReflectedAttributes ahvpset,
                                   String theTag) throws ObjectNotKnown,
                                   AttributeNotKnown, FederateInternalError {

  //This method is invoked when the remote RTIControlModule updates
  //its attributes

  D.dbgOut("Method: CMIFExperimentParticipantFedamb." +
           "reflectAttributeValues()");

  try {

    int size = ahvpset.size();

    // loop through the handle-value pairs in the set

    for (int a = 0; a < size; a++) {

      // obtain the value of the handle

      int ahandle = ahvpset.getHandle(a);

      if (ahandle == parentParticipant.hExperimentState) {
        int tmpInt =
          EncodingHelpers.decodeInt(ahvpset.getValue(a));
      }
      else
        // all we know is population and name, so throw an exception
        throw new AttributeNotKnown("attribute handle is " + ahandle);
    }
  }

  catch(ArrayIndexOutOfBounds ex) {

    /* we shoudn't get this exception, but Java requires us to have a
       handler */

    throw new FederateInternalError("caught array index out of bounds");
```

```
    }
  }//end of reflectAttributeValues
}//end of CMIFExperimentParticipanttFedamb
```

D.3 The package DemoApplication

D.3.1 The Class ContactGenerator.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth        /
//packages hla.rti13.java1      : U.S. DoD               /
//packages java, javax          : Sun Microsystems       /
//-------------------------------------------------------/

package DemoApplication;

import CMIF_HLA_Environment.CMIFExperimentParticipant;

import CMIF_HLA_Environment.CMIFControlCenter.*;
import CMIF_HLA_Environment.util.*;

import hla.rti13.java1.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.swing.*;

//============================================================
//============================================================

public class ContactGenerator
  extends CMIFExperimentParticipant

  implements ActionListener

{

  //This class implements a simple representation of a experiment
  //participant
  //The user can navigate a virtual airplane over a 2D grid. The positions
  //are sent out, as radar contact reports, to the participating
  //TacticDisplays

  DebugHelper D = new DebugHelper();

  JFrame mainWindow;

  GeneratorPanel displayPanel;

  //default display parameters

  public Dimension displayDimension =
    new Dimension(300,300); //display in pixel
  public Rectangle displayArea =
```

```
  new Rectangle(0,0,30000,30000); //real area (in m) which is mapped to display
public Point radarLocation =
  new Point (0,0); //location of the radar sensor
public int radarRange = 0; //radius of the radar reception

public Vector radarContacts = new Vector();

JLabel timeLabel;

JButton clearButton;

JTextField velocityTextField, headingTextField,
  xPosTextField, yPosTextField;

String xPosString = new String("0");
String yPosString = new String("0");
String velocityString = new String("100");
String headingString = new String("45");

double deg2rad = Math.PI/180;

Double xNew, yNew;

//=====================================================
//=====================================================


public static void main(String argv[]) {

  ContactGenerator instance =
    new ContactGenerator(argv);

}//end of main

//=====================================================
//=====================================================

public ContactGenerator(String[] argv) {

  //Constructor of the ContactGenerator Class
  //
  //1) Super Class CMIFExperimentParticipant is instanciated
  //2) other necessary setup methods are kicked off

  super("ContactGenerator");

  D.dbgOut("Started ContactGenerator :");

  if (argv.length>=1) {

    //set values for display dimensions and radar reception
    //(right now, only the default values are used)

  }

  doSetup();

  createDisplay();

  waitForSimulation();

  radarContacts.add(new Point(new Integer(xPosTextField.getText()).intValue(),
```

```
                                    new Integer(yPosTextField.getText()).intValue()));
    startSimulation();

  }//end of constructor

  //=======================================================
  //=======================================================

  private void createDisplay() {

    //This method initializes the display component for the
    //contact generator

    mainWindow = new JFrame(participantName);

    //------------------------------
    //radar display

    displayPanel = new GeneratorPanel(this);

    JPanel centerPanel = new JPanel();
    centerPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    centerPanel.add(displayPanel);

    //------------------------------
    //simulation

    JPanel simulationPanel = new JPanel(new GridLayout(1,0));

    stateIconLabel = new JLabel(orangeIcon);
    timeLabel = new JLabel("--");
    timeLabel.setForeground(Color.black);

    simulationPanel.add(stateIconLabel);
    simulationPanel.add(new JLabel("  "));
    simulationPanel.add(new JLabel("Time:  "));
    simulationPanel.add(timeLabel);
    simulationPanel.setBorder(BorderFactory.createTitledBorder(lineBorder,
                                                    "Simulation"));

    //------------------------------
    //target

    JPanel targetPanel = new JPanel(new GridLayout(0,2));

    velocityTextField = new JTextField(velocityString, 20);
    headingTextField= new JTextField(headingString, 20);
    xPosTextField= new JTextField(xPosString, 20);
    yPosTextField= new JTextField(yPosString, 20);

    targetPanel.add(new JLabel("X Position (m): "));
    targetPanel.add(xPosTextField);
    targetPanel.add(new JLabel("Y Position (m): "));
    targetPanel.add(yPosTextField);
    targetPanel.add(new JLabel("Velocity (m/sec): "));
    targetPanel.add(velocityTextField);
    targetPanel.add(new JLabel("Heading (deg): "));
    targetPanel.add(headingTextField);

    targetPanel.setBorder(BorderFactory.createTitledBorder(lineBorder, "Target"));
```

```
    //------------------------------
    //clear button

    clearButton = new JButton("Clear Display");
    clearButton.addActionListener(this);

    JPanel clearPanel = new JPanel();
    clearPanel.add(clearButton);

    //------------------------------
    //together

    JPanel southPanel = new JPanel(new BorderLayout());
    southPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

    southPanel.add(simulationPanel,BorderLayout.WEST);
    southPanel.add(targetPanel,BorderLayout.EAST);
    southPanel.add(clearPanel,BorderLayout.SOUTH);


    mainWindow.getContentPane().add(centerPanel, BorderLayout.CENTER);
    mainWindow.getContentPane().add(southPanel, BorderLayout.SOUTH);

    mainWindow.pack();
    mainWindow.show();

  }//end of createDisplay

  //=======================================================
  //=======================================================

  public void determineNextPosition() {

    //this method determines the subsequent contact position from
    //the old position, velocity and heading

    D.dbgOut("$" + participantName + ":determineNextPosition()"  );

    double xNow=0.0;
    double yNow= 0.0;
    double velocity=100;
    double heading=0;

    try {

      xNow = (new Double(xPosTextField.getText())).doubleValue();
      yNow = (new Double(yPosTextField.getText())).doubleValue();

      velocity =  (new Double(velocityTextField.getText())).doubleValue();
      heading =  deg2rad*(new Double(headingTextField.getText())).doubleValue();

    } catch(Exception e) {

      D.dbgOut (participantName +
                " :problem with converting numbers from Textfields");

      headingTextField.setText((new Double(heading)).toString());
      velocityTextField.setText((new Double(velocity)).toString());
      headingTextField.revalidate();
      velocityTextField.revalidate();
```

```
    }

    xNew = new Double(xNow + intervalTimeDouble*velocity*Math.sin(heading));
    yNew = new Double(yNow + intervalTimeDouble*velocity*Math.cos(heading));

    xPosTextField.setText(xNew.toString());
    yPosTextField.setText(yNew.toString());

    xPosTextField.revalidate();
    yPosTextField.revalidate();

    radarContacts.add(new Point(xNew.intValue(),
                                yNew.intValue()));


  }//end of determineNextPosition

  //=====================================================
  //=====================================================

  public void sendContactPosition() {

    //This method takes the contact position and reports it
    //to the listening tactic displays

    String positionReport =
      new String("Contact: X=#" + xNew.toString() + "# Y=#" + yNew.toString());

    sendMessageToBus(positionReport, "all");

  }//end of determineNextPosition


  //=====================================================
  //=====================================================
  //=====================================================
  //=====================================================

  //callbacks from the experiment participant

  //=========================================================
  //=========================================================

  public void receivedMessageOnBus(String sender,
                                   String content) {

    //Callback after a message was received
    //..does nothing, since the Contact generator only send out messages

    D.dbgOut("$#" + participantName + " received message from " + sender +
             ", content:" + content);

  }//end of receivedMessageOnBus

  //=========================================================
  //=========================================================

  public void doneWithSetup() {

    //This method is called after the federate is all set up,
    //in contact with the ControlCenter and supplied with neccessary
    //information for the simulation run
```

```
  //Overriding the empty callback

  updateStateIconLabel();

}//end of doneWithSetup

//=========================================================
//=========================================================

public void doneWithShutDown(){

  //This is the callback after the federation is successfully
  //resigned

  D.dbgOut(participantName + ": done with RTI shutdown, now exiting");

  mainWindow.dispose();

  System.exit(0);

}//end of  doneWithShutDown

//=========================================================
//=========================================================

public String reactToPing()
  {

    //A callback after receiving a ping request from the ControlCenter
    //The string that has to be returned will be sent to the ControlCenter
    //as a status message. Other activities can be implemented in
    //overriding this in the participant implementation

    String pingReactionString = new String(participantName +
                                           " is reacting to PING");

    return pingReactionString;

  }//end of reactToPing

//=========================================================
//=========================================================

public void timingParametersChanged() {

  //This is a callback after the ControlCenter has sent
  //timing information (start, stop, scale and increment).
  //The internal processing is already done. This is to notify
  //the implementation of the new situation.

  timeLabel.setText(startTimeString);
  timeLabel.revalidate();


}//end of timingParametersChanged


//=========================================================
//=========================================================

public void startSimulation() {
```

```
    D.dbgOut(participantName + ": startSimulation()");

    updateStateIconLabel();

    while (simIsRunning()) {

      determineNextPosition();

      displayPanel.repaint();

      sendContactPosition();

      advanceOneStep();

    }
  }//end of startSimulation

  //=========================================================
  //=========================================================
  //=========================================================
  //=========================================================

  public void doneWithSimulation(){

    //This is the callback after the simulation is done
    //(=simulation time reached upper limit)

    updateStateIconLabel();

  }//end of  doneWithSimulation

  //=========================================================
  //=========================================================

  public void updateSimTime(double simTime) {

    //This callback can be overriden to do something each time a
    //new simulation time is set...

    timeLabel.setText((new Double(fedamb.mCurrentTime)).toString());

  }//end of updateSimTime


  //=========================================================
  //=========================================================

  public void abortSimulation() {

    D.dbgOut(participantName + ": abortSimulation()");

    updateStateIconLabel();

  }//end of abortSimulation

  //=======================================================
  //=======================================================
  //
  // L I S T E N E R
  //
  //=======================================================
```

```
  public void actionPerformed(ActionEvent event) {

    String command = event.getActionCommand();
    D.dbgOut(participantName +" received: " + command);
    if (command.equals("Clear Display")){
      radarContacts.clear();
      displayPanel.repaint();
    }

  }//end of actionPerformed

}//end of ContactGenerator
```

D.3.2 The Class GeneratorPanel.java

```
//---------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                       /
//                                                         /
//This is part of a Master's Thesis at the                 /
//Center for Multisource Information Fusion                /
//SUNY at Buffalo                                          /
//                                                         /
//All rights reserved:                                     /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth          /
//packages hla.rti13.java1      : U.S. DoD                 /
//packages java, javax          : Sun Microsystems         /
//---------------------------------------------------------/

package CMIF_HLA_Environment.CMIFExperimentParticipant;

import CMIF_HLA_Environment.util.*;


import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.swing.*;
import javax.swing.border.*;

//============================================================
//============================================================

public class GeneratorPanel
  extends JPanel {

  //This class provides the actual display functionalities to
  //the generator panel aplication

  Dimension preferredSize = new Dimension(200, 200);

  DebugHelper D = new DebugHelper(DebugHelper.FULL_DEBUG);

  ContactGenerator parentContactGenerator;

  Border lineBorder = BorderFactory.createLineBorder(Color.black, 2);
  Border emptyBorder = BorderFactory.createEmptyBorder(8,8,8,8);

  Dimension displayDimension ; //display in pixel
  Rectangle displayArea; //real area (in m) which is mapped to display

  int gridDivisions = 5; //the smaller side of the display will be
  //divided into this number of grid elements

  float gridIncrement;

  Color backgroundColor = new Color(0,102,102);
  Color gridColor = new Color(0, 255, 50);

  Color contactColor = new Color(255, 0, 0);

  //============================================================
  //============================================================
```

```
public GeneratorPanel (ContactGenerator parent) {

  //constructor

  D.dbgOut("Started Tactic Display Panel");

  parentContactGenerator = parent;

  displayDimension = parentContactGenerator.displayDimension ;
  displayArea = parentContactGenerator.displayArea;

  setBackground(backgroundColor);
  setForeground(gridColor);

  setBorder(lineBorder);

  //determine grid parameters

  int smallerSide;

  if (displayDimension.height>displayDimension.width) {
    smallerSide=displayDimension.width;
  } else {
    smallerSide=displayDimension.height;
  }

  gridIncrement= smallerSide/gridDivisions;


}//end of constructor

//=========================================================
//=========================================================

public Dimension getPreferredSize() {

  return displayDimension;

}

//=========================================================
//=========================================================

public void paintComponent(Graphics g) {

  //This method does the actual painting work and overrides
  //paintComponent() in JPanel

  super.paintComponent(g);  //paint background

  //paint grid

  g.setColor(gridColor);

  int nxG=1;

  while ((new Float(nxG*gridIncrement)).intValue() <displayDimension.width ){
    g.drawLine((new Float(nxG*gridIncrement)).intValue(),0,
               (new Float(nxG*gridIncrement)).intValue(),
               displayDimension.height);
    nxG++;
```

```
  }

  int nyG=1;

  while ((new Float(nyG*gridIncrement)).intValue() <displayDimension.height ){

    g.drawLine(0,(new Float(nyG*gridIncrement)).intValue(),
               displayDimension.width,
               (new Float(nyG*gridIncrement)).intValue());
    nyG++;

  }

  //paint axes

  int textHeight=12;
  int textWidth=55;

  g.drawString("x=" +(new Integer(displayArea.x)).toString(),
               10,displayDimension.height - textHeight );
  g.drawString("y=" +(new Integer(displayArea.y)).toString(),
               2, displayDimension.height - 2*textHeight);
  g.drawString("y=" +(new Integer(displayArea.y+displayArea.height)).toString(),
               2,textHeight+3);
  g.drawString("x=" +(new Integer(displayArea.x+displayArea.width)).toString(),
               displayDimension.width-textWidth-3,
               displayDimension.height - textHeight);

  //paint contact

  if ( parentContactGenerator.radarContacts.size()>0)
    paintContactVector(g, parentContactGenerator.radarContacts, contactColor );


}//end of paintComponent

//============================================================
//============================================================

private void paintContactVector(Graphics g, Vector tmpVector, Color cColor) {

  //This method paints a little "x" for every contact

  g.setColor(cColor);

  int b = 3;

  for (int i=0;i<tmpVector.size();i++) {

    Point contactC = (Point)tmpVector.get(i);
    Point contactP = c2pix(contactC.x, contactC.y);

    g.drawLine(contactP.x-b, contactP.y-b,contactP.x+b, contactP.y+b );
    g.drawLine(contactP.x-b, contactP.y+b,contactP.x+b, contactP.y-b );

  }

}//end of paintContactVector

//============================================================
//============================================================
```

```
  private Point c2pix(int cX, int cY) {

    //method to convert from "real" coordinate space
    //to pixel coordinate space

    int pX = 0;
    int pY = 0;

    //x-direction

    double xFact = (cX-displayArea.getX())/ displayArea.getWidth();

    pX = (new Float(displayDimension.width*xFact)).intValue();


    //y-direction

    double yFact = (cY-displayArea.getY())/ displayArea.getHeight();

    pY = (new Float(displayDimension.height*(1 - yFact))).intValue();

    return new Point(pX,pY);

  }//end of c2pix

}//end of class GeneratorPanel
```

D.3.3 The Class TacticDisplay.java

```
//---------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                       /
//                                                         /
//This is part of a Master's Thesis at the                 /
//Center for Multisource Information Fusion                /
//SUNY at Buffalo                                          /
//                                                         /
//All rights reserved:                                     /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth          /
//packages hla.rti13.java1      : U.S. DoD                 /
//packages java, javax          : Sun Microsystems         /
//---------------------------------------------------------/

package DemoApplication;

import CMIF_HLA_Environment.CMIFExperimentParticipant;
import CMIF_HLA_Environment.CMIFControlCenter.*;
import CMIF_HLA_Environment.util.*;

import hla.rti13.java1.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.swing.*;

//============================================================
//============================================================

public class TacticDisplay
  extends CMIFExperimentParticipant

  implements ActionListener

{

  //This class implements a simple representation of a
  //tactic display
  //"Radar" information is received, displayed and exchanged with other
  //tactic displays

  DebugHelper D = new DebugHelper();

  JFrame mainWindow;

  TacticDisplayPanel displayPanel;

  public Dimension displayDimension =
    new Dimension(300, 200); //display in pixel

  public Rectangle displayArea =
    new Rectangle(0,0,30000,20000); //real area (in m) which is mapped to display

  public Point radarLocation =
    new Point (10000, 10000); //location of the radar sensor

  public int radarRange = 7000; //radius of the radar reception
```

```
public Vector radarContacts = new Vector();
public Vector reportedContacts = new Vector();

JLabel timeLabel;

JButton clearButton;

JCheckBox broadcastCheckBox, receiveCheckBox;

//=======================================================
//=======================================================


public static void main(String argv[]) {

  TacticDisplay instance =
    new TacticDisplay(argv);

}//end of main

//=======================================================
//=======================================================

public TacticDisplay(String[] argv) {

  //Constructor of the TacticDisplay Class
  //
  //1) Super class CMIFExperimentParticipant is instanciated
  //2) Display Parameters are read from the command line input
  //3) Remaining setup methods are called

  super(argv[0]);

  D.dbgOut("Started TacticDisplay :" + argv[0]);

  try {

    displayArea =
      new Rectangle((new Integer(argv[1])).intValue(),
                    (new Integer(argv[2])).intValue(),
                    (new Integer(argv[3])).intValue(),
                    (new Integer(argv[4])).intValue());

    radarLocation = new Point ((new Integer(argv[5])).intValue(),
                               (new Integer(argv[6])).intValue());

    radarRange = (new Integer(argv[7])).intValue();

    //map the displayArea (in m) to a reasonable screen represenation
    //where the longest side of the display area is limited
    //to dispMax

    int dispMax = 400;

    double dispRatio = displayArea.getWidth()/displayArea.getHeight();

    D.dbgOut("Display Ratio:  " + (new Double(dispRatio)).toString());
    D.dbgOut("Display Area:  " + displayArea.toString());

    if (dispRatio>=1) {
```

```
        displayDimension.width = dispMax;
        displayDimension.height = (new Double(dispMax/dispRatio)).intValue();

      } else {

        displayDimension.height = dispMax;
        displayDimension.width = (new Double(dispMax*dispRatio)).intValue();

      }

      D.dbgOut(participantName + ": Setting pixel display to " +
               displayDimension);

    } catch (Exception e) {

      D.dbgOut("Usage: ... <name> <xLL> <yLL> <Width> <Height> <radX> "+
               "<radY> <radRange>");
      D.dbgOut("#<xLL>, <yLL>: Location of the lower left corner of the "+
               "display area (in m)");
      D.dbgOut("#<Width>, <Height>: Width and Height of the display in m");
      D.dbgOut("#<radX>, <radY>: Location of the radar sensor");
      D.dbgOut("#<radRange>: Range of the radar sensor");

    }


    doSetup();

    createDisplay();

    waitForSimulation();

    startSimulation();

  }//end of constructor

  //=========================================================
  //=========================================================

  private void createDisplay() {

    //This method initializes the display component for the
    //tactical display

    mainWindow = new JFrame(participantName);

    //-----------------------------
    //radar display

    displayPanel = new TacticDisplayPanel(this);

    JPanel centerPanel = new JPanel();
    centerPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    centerPanel.add(displayPanel);

    //-----------------------------
    //simulation

    JPanel simulationPanel = new JPanel(new GridLayout(1,0));
```

```
    stateIconLabel = new JLabel(orangeIcon);
    timeLabel = new JLabel("--");
    timeLabel.setForeground(Color.black);

    simulationPanel.add(stateIconLabel);
    simulationPanel.add(new JLabel("  "));
    simulationPanel.add(new JLabel("Time:  "));
    simulationPanel.add(timeLabel);
    simulationPanel.setBorder(BorderFactory.createTitledBorder(lineBorder,
                                                        "Simulation"));
    //-----------------------------
    //communications

    JPanel commPanel = new JPanel(new GridLayout(1,0));

    broadcastCheckBox = new JCheckBox("Broadcast Contacts",true);
    receiveCheckBox = new JCheckBox("Receive Contacts",true);

    commPanel.add(broadcastCheckBox);
    commPanel.add(receiveCheckBox);
    commPanel.setBorder(BorderFactory.createTitledBorder(lineBorder,
                                                    "Communications"));

    //-----------------------------
    //clear button

    clearButton = new JButton("Clear Display");
    clearButton.addActionListener(this);

    JPanel clearPanel = new JPanel();
    clearPanel.add(clearButton);

    //-----------------------------
    //together

    JPanel southPanel = new JPanel(new BorderLayout());
    southPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

    southPanel.add(simulationPanel,BorderLayout.WEST);
    southPanel.add(clearPanel,BorderLayout.SOUTH);
    southPanel.add(commPanel,BorderLayout.EAST);


    mainWindow.getContentPane().add(centerPanel, BorderLayout.CENTER);
    mainWindow.getContentPane().add(southPanel, BorderLayout.SOUTH);

    mainWindow.pack();
    mainWindow.show();


}//end of createDisplay


//=================================================
//=================================================

private boolean evaluateRadarRange(Point contactPoint){

  //This method tests for a given contactPoint if this
  //point is within the radar range of this tactic display
```

```
    double vecSq = (new Integer((contactPoint.x-radarLocation.x)*
                                (contactPoint.x-radarLocation.x) +
                                (contactPoint.y-radarLocation.y)*
                                (contactPoint.y-radarLocation.y))).doubleValue();

    boolean withinRange =
      (Math.sqrt(vecSq) < (new Double(radarRange)).intValue());

    D.dbgOut("$#Contact has distance from radar: " +
             (new Double(Math.sqrt(vecSq))).toString() );

    return withinRange;

  }//end of evaluateRadarRange

  //=======================================================
  //=======================================================

  //callbacks from the experiment participant

  //=============================================================
  //=============================================================

  //=============================================================
  //=============================================================

  public void startSimulation() {

    D.dbgOut(participantName + ": startSimulation()");

    //stateIconLabel.setIcon(greenIconPulse);
    //stateIconLabel.revalidate();

    updateStateIconLabel();

    while (simIsRunning()) {

      displayPanel.repaint();

      advanceOneStep();

    }

  }//end of startSimulation

  //=============================================================
  //=============================================================

  public void receivedMessageOnBus(String sender,
                                   String content) {

    //This Method evaluates the incoming message which
    //is either a "contact" or a "reported contact".
    //A contact Point is generated and added to the corresponding
    //contactVector.
    //If it is an genuine contact, also the rebroadcast is issued

    String delim = "#";

    StringTokenizer messageTokenizer =
      new StringTokenizer(content, delim);
```

```
        String head = messageTokenizer.nextToken();
        String xString = messageTokenizer.nextToken();
        messageTokenizer.nextToken();
        String yString = messageTokenizer.nextToken();

        Double xC = new Double(xString);
        Double yC = new Double(yString);

        Point contactPoint = new Point (xC.intValue(), yC.intValue());

        if (head.startsWith("Contact")){

          boolean isWithinRange = evaluateRadarRange(contactPoint);

          if (isWithinRange) {

            radarContacts.add(contactPoint);

            if (broadcastCheckBox.isSelected())
              sendMessageToBus("Reported"+content, "all");

          }

        }


        if (head.startsWith("Reported") && receiveCheckBox.isSelected())
          reportedContacts.add(contactPoint);

      }//end of receivedMessageOnBus

      //============================================================
      //============================================================

      public void doneWithSetup() {

        //This method is called after the federate is all set up,
        //in contact with the ControlCenter and supplied with neccessary
        //information for the simulation run

        //...in this case, only the state icon is switched to green

        updateStateIconLabel();

      }//end of doneWithSetup

      //============================================================
      //============================================================

      public void abortSimulation() {

        //This is a callback after the ControlCenter has sent
        //the signal to abort the simulation

        //...in this case, only the state icon is switched to red

        updateStateIconLabel();


      }//end of startSimulation
```

```
//=========================================================
//=========================================================

public void doneWithSimulation(){

  //This is the callback after the simulation is over (=reached
  //upper time limit)

  updateStateIconLabel();

}//end of  doneWithSimulation

//=========================================================
//=========================================================

public void doneWithShutDown(){

  //This is the callback after the federation is successfully
  //resigned

  D.dbgOut(participantName + ": done with RTI shutdown, now exiting");

  mainWindow.dispose();

  System.exit(0);

}//end of  doneWithShutDown

//=========================================================
//=========================================================

public String reactToPing()
  {

    //A callback after receiving a ping request from the ControlCenter
    //The string that has to be returned will be sent to the ControlCenter
    //as a status message. Other activities can be implemented in
    //overriding this in the participant implementation

    String pingReactionString = new String(participantName +
                                           " is reacting to PING");

    return pingReactionString;

  }//end of reactToPing

//=========================================================
//=========================================================

public void timingParametersChanged() {

  //This is a callback after the ControlCenter has sent
  //timing information (start, stop, scale and increment).
  //The internal processing is already done. This is to notify
  //the implementation of the new situation.

  timeLabel.setText(startTimeString);
  timeLabel.revalidate();

}//end of timingParametersChanged

//=========================================================
```

```
  //=======================================================

  public void updateSimTime(double simTime) {

    //This callback can be overriden to do something each time a new simulation
    //time is set...

    timeLabel.setText((new Double(simTime)).toString());

  }//end of updateSimTime

  //=======================================================
  //=======================================================
  //
  // L I S T E N E R
  //
  //=======================================================

  public void actionPerformed(ActionEvent event) {

    String command = event.getActionCommand();

    D.dbgOut(participantName +" received: " + command);

    if (command.equals("Clear Display")){

      radarContacts.clear();
      reportedContacts.clear();

      displayPanel.repaint();
    }

  }//end of actionPerformed

}//end of class TacticDisplay
```

D.3.4 The Class TacticDisplayPanel.java

```
//---------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                       /
//                                                         /
//This is part of a Master's Thesis at the                 /
//Center for Multisource Information Fusion                /
//SUNY at Buffalo                                          /
//                                                         /
//All rights reserved:                                     /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth          /
//packages hla.rti13.java1      : U.S. DoD                 /
//packages java, javax          : Sun Microsystems         /
//---------------------------------------------------------/

package DemoApplication;

import CMIF_HLA_Environment.CMIFExperimentParticipant;
import CMIF_HLA_Environment.util.*;


import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.swing.*;
import javax.swing.border.*;

//============================================================
//============================================================

public class TacticDisplayPanel
  extends JPanel {

  //This class implements the display painting functions
  //needed for the TacticDisplay application

  Dimension preferredSize = new Dimension(200, 200);

  DebugHelper D = new DebugHelper(DebugHelper.FULL_DEBUG);

  TacticDisplay parentTacticDisplay;

  Border lineBorder = BorderFactory.createLineBorder(Color.black, 2);
  Border emptyBorder = BorderFactory.createEmptyBorder(8,8,8,8);

  Dimension displayDimension ; //display in pixel
  Rectangle displayArea; //real area (in m) which is mapped to display
  Point radarLocation; //location of the radar sensor
  int radarRange; //radius of the radar reception

  int gridDivisions = 5; //the smaller side of the display will be
  //divided into this number of grid elements

  float gridIncrement;

  Color backgroundColor = new Color(0,102,102);
  Color gridColor = new Color(0, 255, 50);

  Color radarCircleColor = new Color(255, 255, 0);
```

```
Color contactColor = new Color(255, 0, 0);
Color repContactColor = Color.gray;

//=========================================================
//=========================================================

public TacticDisplayPanel (TacticDisplay parent) {

  //constructor

  D.dbgOut("Started Tactic Display Panel");

  parentTacticDisplay = parent;

  //get display dimensions from parent

  displayDimension = parentTacticDisplay.displayDimension ;
  displayArea = parentTacticDisplay.displayArea;
  radarLocation = parentTacticDisplay.radarLocation;
  radarRange = parentTacticDisplay.radarRange;

  setBackground(backgroundColor);
  setForeground(gridColor);

  setBorder(lineBorder);

  //determine grid parameters

  int smallerSide;

  if (displayDimension.height>displayDimension.width) {
    smallerSide=displayDimension.width;
  } else {
    smallerSide=displayDimension.height;
  }

  gridIncrement= smallerSide/gridDivisions;


}//end of constructor

//=========================================================
//=========================================================

public Dimension getPreferredSize() {

  return displayDimension;

}

//=========================================================
//=========================================================

public void paintComponent(Graphics g) {

  //This method overrides paintComponent in JPanel
  //and does the actual painting work

  super.paintComponent(g);  //paint background

  //paint grid
```

```
g.setColor(gridColor);

int nxG=1;

while ((new Float(nxG*gridIncrement)).intValue() <displayDimension.width ){
  g.drawLine((new Float(nxG*gridIncrement)).intValue(),0,
             (new Float(nxG*gridIncrement)).intValue(),
             displayDimension.height);
  nxG++;
}

int nyG=1;

while ((new Float(nyG*gridIncrement)).intValue() <displayDimension.height ){

  g.drawLine(0,(new Float(nyG*gridIncrement)).intValue(),
             displayDimension.width,
             (new Float(nyG*gridIncrement)).intValue());
  nyG++;

}

//paint axes

int textHeight=12;
int textWidth=55;

g.drawString("x=" +(new Integer(displayArea.x)).toString(),
             10,displayDimension.height - textHeight );
g.drawString("y=" +(new Integer(displayArea.y)).toString(),
             2, displayDimension.height - 2*textHeight);
g.drawString("y=" +(new Integer(displayArea.y+displayArea.height)).toString(),
             2,textHeight+3);
g.drawString("x=" +(new Integer(displayArea.x+displayArea.width)).toString(),
             displayDimension.width-textWidth-3,displayDimension.height -
             textHeight);

//paint radar range

g.setColor(radarCircleColor);

Point radarCenter = c2pix(radarLocation.x, radarLocation.y);

g.fillOval(radarCenter.x-3, radarCenter.y-3, 6, 6);

Point radarUpperLeft = c2pix(radarLocation.x-radarRange,
                             radarLocation.y+radarRange);

Point radarLowerRight = c2pix(radarLocation.x+radarRange,
                              radarLocation.y-radarRange);

g.drawOval(radarUpperLeft.x, radarUpperLeft.y,
           radarLowerRight.x-radarUpperLeft.x,
           radarLowerRight.y-radarUpperLeft.y);

//paint contact

if ( parentTacticDisplay.radarContacts.size()>0)
  paintContactVector(g, parentTacticDisplay.radarContacts, contactColor );

if ( parentTacticDisplay.reportedContacts.size()>0)
```

```
    paintReportedContactVector(g, parentTacticDisplay.reportedContacts,
                               repContactColor );

}//end of paintComponent

//=========================================================
//=========================================================

private void paintContactVector(Graphics g, Vector tmpVector, Color cColor) {

  //...paints a little "x" for every entry in the contact vector...

  g.setColor(cColor);

  int b = 3;

  for (int i=0;i<tmpVector.size();i++) {

    Point contactC = (Point)tmpVector.get(i);
    Point contactP = c2pix(contactC.x, contactC.y);

    g.drawLine(contactP.x-b, contactP.y-b,contactP.x+b, contactP.y+b );
    g.drawLine(contactP.x-b, contactP.y+b,contactP.x+b, contactP.y-b );

  }

}//end of paintContactVector

//=========================================================
//=========================================================

private void paintReportedContactVector(Graphics g,
                                        Vector tmpVector,
                                        Color cColor) {

  //....paints a little "+" for every entry in the reported
  //contacts vector

  g.setColor(cColor);

  int b = 3;

  for (int i=0;i<tmpVector.size();i++) {

    Point contactC = (Point)tmpVector.get(i);
    Point contactP = c2pix(contactC.x, contactC.y);

    g.drawLine(contactP.x, contactP.y-b,contactP.x, contactP.y+b );
    g.drawLine(contactP.x-b, contactP.y,contactP.x+b, contactP.y );

  }

}//end of paintReportedContactVector

//=========================================================
//=========================================================

private Point c2pix(int cX, int cY) {

  //method to convert from "real" coordinate space
  //to pixel coordinate space
```

```
    int pX = 0;
    int pY = 0;

    //x-direction

    double xFact = (cX-displayArea.getX())/ displayArea.getWidth();

    pX = (new Float(displayDimension.width*xFact)).intValue();

    //y-direction

    double yFact = (cY-displayArea.getY())/ displayArea.getHeight();

    pY = (new Float(displayDimension.height*(1 - yFact))).intValue();

    return new Point(pX,pY);

  }//end of c2pix
}//end of class TacticDisplayPanel
```

D.4 The package util

D.4.1 The Class DebugHelper.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This code is part of a Master's Thesis at the          /
//Center for Multisource Information Fusion,             /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth        /
//packages hla.rti13.java1      : U.S. DoD               /
//packages java, javax          : Sun Microsystems       /
//-------------------------------------------------------/

package CMIF_HLA_Environment.util;

//=========================================================
//=========================================================

public class DebugHelper

{
  //The class DebugHelper facilitates the coordinated output
  //of debug messages throughout the CMIF_HLA_Environment

  public static int FULL_DEBUG  = 1;
  public static int LIGHT_DEBUG = 2;
  public static int NO_DEBUG = 3;

  int debugState;

  //=====================================================
  //=====================================================

  public DebugHelper (){

    debugState = LIGHT_DEBUG;

  }//end of contructor

  //=====================================================
  //=====================================================

  public DebugHelper (int dState){

    if ((dState>=LIGHT_DEBUG)||(dState<=NO_DEBUG)) {

      debugState = dState;

    } else {

      debugState = LIGHT_DEBUG;

    }

  }//end of contructor
```

```
//=======================================================
//=======================================================

public void dbgOut(String message) {

  //This method writes the given message to System.err
  //if the prefix matches the current debug state:
  //- no prefix  : display always if debug is on,
  //- "$" prefix : display only when debug is FULL_DEBUG
  //- "#" prefix : no separation lines in between

  if (debugState!=NO_DEBUG) {

    boolean fullDebugMessage = false;
    boolean lineSeparation = true;

    if (message.startsWith("$")) {

      fullDebugMessage = true;
      message = message.substring(1);

    }

    if (message.startsWith("#")) {

      lineSeparation = false;
      message = message.substring(1);

    }

    if ((debugState==LIGHT_DEBUG) && (!fullDebugMessage)) {

      if (lineSeparation){
        System.err.println("-------------------------");
        System.err.println("");
      }

      System.err.println(message);

    } else {

      if (lineSeparation){
        System.err.println("-------------------------");
        System.err.println("");
      }

      System.err.println(message);

    }

  }

}//end of dbgOut()

//=======================================================
//=======================================================

public void setDebugState (int newdState) {

  //This method allows to change the debugState during runtime

  if ((newdState>=LIGHT_DEBUG)||(newdState<=NO_DEBUG)) {
```

```
      debugState = newdState;
    }
  }//end of setDebugState()

}//end of DebugHelper
```

D.4.2 The Class ExperimentFileHandler.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                    : CMIF, Kai Harth        /
//packages hla.rti13.java1      : U.S. DoD               /
//packages java, javax          : Sun Microsystems       /
//-------------------------------------------------------/

package CMIF_HLA_Environment.util;


import CMIF_HLA_Environment.CMIFControlCenter.*;

import javax.swing.*;
import java.io.*;
import java.awt.event.*;
import java.util.*;

//============================================================
//============================================================

public class ExperimentFileHandler

{

  //The class ExperimentFileHandler performs the tasks of loading and saving
  //files containing experiment setup data.
  //
  //Some features of the .cef file remain unused in this version but will be
  //used later.
  //
  //In use right now: timing info, experiment ino and participants list

  public static String fileEnding = ".cef"; //stands for CMIF Experiment File

  //============================================================
  //============================================================


  public static void loadExperimentFile(CMIFExperimentManager manager) {

    manager.D.dbgOut("$Method: ExperimentFileHandler.loadExperimentFile");

    try {

      String fileSeparator =
        new String( System.getProperty("file.separator"));

      String userDirectory =
        new String( System.getProperty("user.dir") + fileSeparator);

      JFileChooser fileChooser = new JFileChooser( userDirectory );
```

```
    fileChooser.setFileFilter(new javax.swing.filechooser.FileFilter() {
      public boolean accept(File f) {
        boolean acc = false;
        if (f.isDirectory())
          acc = true;
        if(f.getName().endsWith("cef"))
          acc = true;
        return acc;
      }
      public String getDescription() {
        return new String("CMIF Experiment File");
      }
    }
                              );
    int returnVal = fileChooser.
      showOpenDialog(manager.parentCCenter.CCDisplay);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
      File file = fileChooser.getSelectedFile();
      FileInputStream in = new
        FileInputStream(file);
      byte bt[] = new byte[(int)file.length()];
      in.read(bt);
      String contentString = new String(bt);
      in.close();
      analyzeFileContent(contentString, manager);
    }
  } catch (Exception e) {
    manager.
      parentCCenter.CCDisplay.messageOut("Loading failed: " + e.toString());
  }

}//end of loadExperimentFile

//============================================================
//============================================================

public static void saveExperimentFile(CMIFExperimentManager manager) {
  manager.D.dbgOut("$Method: ExperimentFileHandler.saveExperimentFile");
  try {
    String fileSeparator =
      new String( System.getProperty("file.separator"));
```

```
String userDirectory =
  new String( System.getProperty("user.dir") + fileSeparator);

String fileName =
  new String(userDirectory + manager.experimentNameString +
             fileEnding);
String title =
  new String("Save Experiment Data");
JPanel messagePanel =
  new JPanel();
messagePanel.setLayout(new BoxLayout(messagePanel,BoxLayout.Y_AXIS));

messagePanel.add(new JLabel("Save data to:"));
messagePanel.add(new JLabel(fileName));


int returnInt = JOptionPane.
  showConfirmDialog((JFrame)manager.parentCCenter.CCDisplay,
                    messagePanel,
                    title,
                    JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.QUESTION_MESSAGE);

if (returnInt==JOptionPane.OK_OPTION) {

  manager.
    parentCCenter.CCDisplay.messageOut("Saving...");

  FileWriter experimentFileWriter =
    new FileWriter(fileName);

  experimentFileWriter.write("#======================#\n");
  experimentFileWriter.write("# CMIF_HLA_ENVIROMENT   #\n");
  experimentFileWriter.write("# Experiment Data File  #\n");
  experimentFileWriter.write("#======================#\n");
  experimentFileWriter.write("\n");
  experimentFileWriter.write("#Generated File! Edit only if you know " +
                             "what you are doing....\n");
  experimentFileWriter.write("\n");
  experimentFileWriter.write("# general info\n");
  experimentFileWriter.write("\n");
  experimentFileWriter.write(">>author{" +
                             manager.experimentAuthorString+"}\n");
  experimentFileWriter.write("\n");
  experimentFileWriter.write(">>name{" +
                             manager.experimentNameString +"}");
  experimentFileWriter.write("\n");
  experimentFileWriter.write(">>description{" +
                             manager.experimentDescriptionString +"}");
  experimentFileWriter.write("\n\n");
  experimentFileWriter.write("# time management\n");
  experimentFileWriter.write("\n");
  experimentFileWriter.write(">>start_time{" +
                             manager.startTimeString +"}\n");
  experimentFileWriter.write(">>end_time{" +
                             manager. endTimeString +"}\n");
  experimentFileWriter.write(">>interval_time{" +
                             manager.intervalTimeString +"}\n");
```

```
        experimentFileWriter.write(">>scale{" +
                                   manager.scaleString +"}\n");

        experimentFileWriter.write("\n\n");
        experimentFileWriter.write("# participants\n");
        experimentFileWriter.write("\n");

        int participantsNum = manager.participantsToJoinVector.size();

        experimentFileWriter.write(">>participant_num{" +participantsNum +
                                   "}\n");

        for (int i=0; i< participantsNum; i++) {

          String tmpString =
            (String)manager.participantsToJoinVector.elementAt(i);

          experimentFileWriter.write(">>participant{" + (i+1) +", " +
                                     tmpString +"}\n");
        }


        experimentFileWriter.write("\n\n");
        experimentFileWriter.write("# channels\n");
        experimentFileWriter.write("\n");

        experimentFileWriter.close();

      }

    } catch (Exception e) {

      manager.
        parentCCenter.CCDisplay.messageOut("Saving failed: " + e.toString());

    }

}//end of saveExperimentFile

//============================================================
//============================================================

private static void analyzeFileContent(String wholeFileString,
                                       CMIFExperimentManager manager) {


  //This method breaks down the string read from the experiment
  //data file (at the ">>" delimiters), analyzes the information
  //and puts it back in place for the experiment manager..

  String delim = new String(">>");

  StringTokenizer wholeFileTokenizer =
    new StringTokenizer(wholeFileString, delim);

  Vector tmpVector = new Vector();


  //skip the file header
  if (wholeFileTokenizer.hasMoreTokens()){
    String dump = wholeFileTokenizer.nextToken();
```

```
                                }
    while (wholeFileTokenizer.hasMoreTokens()) {

      StringTokenizer segmentTokenizer =
        new StringTokenizer(wholeFileTokenizer.nextToken(),"{}");

      if (segmentTokenizer.countTokens() >1) {

        String keyString = segmentTokenizer.nextToken();
        String valueString = segmentTokenizer.nextToken();


        if (keyString.equals("name")) {

          manager.experimentNameString = valueString;
          manager.experimentNameTF.setText(valueString);
          manager.parentCCenter.CCDisplay.infoLabel.setText(valueString);

        } else if (keyString.equals("description")) {

          manager.experimentDescriptionString = valueString;
          manager.experimentInfoTA.setText(valueString);

        } else if (keyString.equals("author")) {

          manager.experimentAuthorString = valueString;
          manager.experimentAuthorTF.setText(valueString);

        } else if (keyString.equals("start_time")) {

          manager.startTimeString = valueString;
          manager.parentCCenter.CCDisplay.startTimeTF.setText(valueString);

        } else if (keyString.equals("end_time")) {

          manager.endTimeString = valueString;
          manager.parentCCenter.CCDisplay.endTimeTF.setText(valueString);

        } else if (keyString.equals("interval_time")) {

          manager.intervalTimeString = valueString;
          manager.parentCCenter.CCDisplay.intervalTF.setText(valueString);

        } else if (keyString.equals("scale")) {

          manager.scaleString = valueString;
          manager.parentCCenter.CCDisplay.scaleTF.setText(valueString);

        } else if (keyString.equals("participant_num")) {

          manager.participantsToJoinVector.clear();
          manager.participantsToJoinVector.
            setSize((new Integer(valueString)).intValue());

          manager.D.dbgOut("$#Setting capacity of vector to: " +
                           (new Integer(valueString)).intValue());

        } else if (keyString.equals("participant")) {

          StringTokenizer participantTokenizer =
            new StringTokenizer(valueString,", ");
```

```
            int posInt = (new Integer(participantTokenizer.nextToken())).
              intValue();

            String tmpString = participantTokenizer.nextToken();

            manager.D.dbgOut("$#Setting #" + (posInt-1) +" to " + tmpString );

            manager.participantsToJoinVector.
              setElementAt(tmpString, posInt-1);

          } else {

            manager.parentCCenter.
              CCDisplay.messageOut("");
            manager.parentCCenter.
              CCDisplay.messageOut("An error occured while reading "+
                                   "the data file!");
            manager.parentCCenter.
              CCDisplay.messageOut("The file might be corrupted....");

            manager.parentCCenter.
              CCDisplay.messageOut( keyString + ", " + valueString);

          }

        }

      }//end of while

      manager.evaluateTimePanelInputs();

      manager.participantsConfigList.setListData(manager.participantsToJoinVector);

      manager.participantsConfigList.validate();

      manager.parentCCenter.cCenterState=CMIFControlCenter.SETUP_EXPERIMENT;
      manager.parentCCenter.CCDisplay.setPhaseIcons("config", "grn_pulse");
      manager.parentCCenter.CCDisplay.setPhaseIcons("simulation", "red");
      manager.parentCCenter.CCDisplay.setPhaseIcons("rticleanup", "red");
      manager.parentCCenter.CCDisplay.setPhaseIcons("shutdown", "orange");

    }//end of analyzeFileContentnt

};//end of ExperimeFileHandler
```

D.4.3 The Class SwingWorker.java

```
//-------------------------------------------------------/
//CMIF_HLA_Environment v1.00, 7/2000                     /
//                                                       /
//This is part of a Master's Thesis at the               /
//Center for Multisource Information Fusion              /
//SUNY at Buffalo                                        /
//                                                       /
//All rights reserved:                                   /
//packages CMIFControlCenter, CMIFExperimentParticipant,/
//      util                  : CMIF, Kai Harth          /
//packages hla.rti13.java1    : U.S. DoD                 /
//packages java, javax        : Sun Microsystems         /
//-------------------------------------------------------/

//>>> This class is adapted from the Java Tutorial at java.sun.com

package CMIF_HLA_Environment.util;

import javax.swing.SwingUtilities;

/**
 * This is the 3rd version of SwingWorker (also known as
 * SwingWorker 3), an abstract class that you subclass to
 * perform GUI-related work in a dedicated thread.  For
 * instructions on using this class, see:
 *
 * http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html
 *
 * Note that the API changed slightly in the 3rd version:
 * You must now invoke start() on the SwingWorker after
 * creating it.
 */

//=========================================================
//=========================================================

public abstract class SwingWorker {

  private Object value;  // see getValue(), setValue()
  private Thread thread;

  /**
   * Class to maintain reference to current worker thread
   * under separate synchronization control.
   */

  //=========================================================
  //=========================================================

  private static class ThreadVar {
    private Thread thread;
    ThreadVar(Thread t) { thread = t; }
    synchronized Thread get() { return thread; }
    synchronized void clear() { thread = null; }
  }

  //=========================================================
  //=========================================================
```

```
private ThreadVar threadVar;

/**
 * Get the value produced by the worker thread, or null if it
 * hasn't been constructed yet.
 */

//=============================================================
//=============================================================

protected synchronized Object getValue() {
  return value;
}

//=============================================================
//=============================================================

/**
 * Set the value produced by worker thread
 */

private synchronized void setValue(Object x) {
  value = x;
}

//=============================================================
//=============================================================

/**
 * Compute the value to be returned by the <code>get</code> method.
 */

public abstract Object construct();

//=============================================================
//=============================================================

/**
 * Called on the event dispatching thread (not on the worker thread)
 * after the <code>construct</code> method has returned.
 */

public void finished() {

}

//=============================================================
//=============================================================

/**
 * A new method that interrupts the worker thread.  Call this method
 * to force the worker to stop what it's doing.
 */

public void interrupt() {
  Thread t = threadVar.get();
  if (t != null) {
    t.interrupt();
  }
  threadVar.clear();
}
```

```
  //=========================================================
  //=========================================================

  /**
   * Return the value created by the <code>construct</code> method.
   * Returns null if either the constructing thread or the current
   * thread was interrupted before a value was produced.
   *
   * @return the value created by the <code>construct</code> method
   */

  public Object get() {
    while (true) {
      Thread t = threadVar.get();
      if (t == null) {
return getValue();
      }
      try {
t.join();
      }
      catch (InterruptedException e) {
Thread.currentThread().interrupt(); // propagate
return null;
      }
    }
  }

  //=========================================================
  //=========================================================

  /**
   * Start a thread that will call the <code>construct</code> method
   * and then exit.
   */

  public SwingWorker() {

    final Runnable doFinished = new Runnable() {
      public void run() { finished(); }

    };

    Runnable doConstruct = new Runnable() {
      public void run() {
try {
  setValue(construct());
}
finally {
  threadVar.clear();
}

SwingUtilities.invokeLater(doFinished);
      }
    };

    Thread t = new Thread(doConstruct);
    threadVar = new ThreadVar(t);
  }

  //=========================================================
  //=========================================================
```

```
  /**
   * Start the worker thread.
   */
  public void start(int priority) {
    Thread t = threadVar.get();
    if (t != null) {
      t.setPriority(priority);
      t.start();
    }
  }

  //=============================================================
  //=============================================================

  public void start() {
    Thread t = threadVar.get();
    if (t != null) {
      t.start();
    }
  }

}//end of SwingWorker
```

Bibliography

[Cas00] *CASE_ATTI, A Testbed for Maritime Data Fusion (Technical Bulletin).* Defense Research Establishment Valcartier, Canada, 2000. http://-www.drev.dnd.ca/tech/marinfus_e.html.

[Def00a] DMSO Software Download Center, December 2000. http://sdc.dmso.mil.

[Def00b] *RTI 1.3–Next Generation Programmer's Guide Version 3.1.* Defense Modeling and Simulation Office, 2000. http://sdc.dmso.mil.

[Def00c] *RTI 1.3–NG Programmer's Guide Version 3.1, Appendix B: FederateAmbassador.* Defense Modeling and Simulation Office, 2000. http://sdc.dmso.mil.

[Def00d] *RTI 1.3–NG Programmer's Guide Version 3.1, Appendix C: Classes and Suporting Types.* Defense Modeling and Simulation Office, 2000. http://sdc.dmso.mil.

[Def00e] *RTI 1.3–NG Programmer's Guide Version 3.1, Appendix A: RTIAmbassador.* Defense Modeling and Simulation Office, 2000. http://sdc.dmso.mil.

[Def00f] *RTI 1.3–NG Version 3.1, Installation Guide.* Defense Modeling and Simulation Office, 2000. http://sdc.dmso.mil.

[Def00g] Website of the Defense Modeling and Simulation Office, December 2000. http://www.dmso.mil.

[Jav00a] Java 2 Platform, "*A Practical Guide for Programmers*", Online version, December 2000. http://java.sun.com/docs/books/tutorial.

[Jav00b] Java 2 Platform, Standard Edition, v1.2.2 API Specification, December 2000. http://java.sun.com/products/jdk/1.2/docs/api/index.html.

[JMP+99] Judith Dahmann, Marnie Salisbury, Phil Barry, et al. HLA and Beyond: Interoperability Challenges . In *Information & Security. An International Journal, Vol. 3*, pages 25–42, 1999.

[MK98] Mary Campione and Kathy Walrath. *The Java Tutorial.* Addison-Wesley, New York, 1998.

[MK99] Mary Campione and Kathy Walrath. *The JFC Swing Tutorial: A Guide to Constructing GUIs.* Addison-Wesley, New York, 1999.

[SDG98] SDG Standard Development Group. *High Level Architecture Rules, Version 1.3.* Defense Modeling and Simulation Office, 1998.